Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Luca Bettermann

Design Exploration of Acoustic Panels with Conditional Variational Autoencoders

Master Thesis

Swiss Data Science Center
Department of Mechanical and Process Engineering
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

Prof. Dr. Mirko Meboldt
Prof. Dr. Fernando Perez-Cruz
Dr. Luis Salamanca

July 2022

Abstract

Designers and engineers are often confronted with problems that involve a large number of parameters, contradicting objectives and a set of various constraints. Finding solutions in this high-dimensional setting that need to live up to certain requirements is a time-consuming procedure. In this thesis a deep learning concept is proposed which enables the reversal of this process by learning the underlying relationship of input parameters and performance attributes. This empowers the user to request a set of attributes and generate various design instances meeting those requirements. The model augments the users ability to efficiently explore the solution space of the given design problem. In order to achieve this objective, a Conditional Variational Autoencoder is employed. This concept is demonstrated on the generation of acoustic panel designs based on requested performance attributes that describe geometrical and functional properties of the panels. Additionally, an analysis framework is developed, enabling the model to evaluate its own performance by estimating the attributes of the generated panels. The models performance is enhanced by implementing a multitude of transformations and enriching the dataset with supplementary data representations. In order to enable an accurate and diverse generation of conditioned acoustic panels, a novel method that encourages the model to decouple the latent space from the conditionality is developed. Lastly, a use case is presented guiding the reader through a practical application of the generative model.

Preface

First and foremost, I would like to thank Dr. Luis Salamanca for the exceptional guidance of this thesis. I am grateful for the time and energy that Luis commited to support me whenever I needed assistance, and for the many insightful discussions we had over the course of the last six months. I enjoyed the remote conversations from the living room, the mountains, or different countries, but even more so in person at the SDSC. It was a pleasure to work with and learn from you. I would like to thank Prof. Dr. Fernando Perez-Cruz for the trust of carrying out this thesis at the SDSC and letting a D-MAVT student into the building. Finally, a big thank you to my tutor Prof. Dr. Mirko Meboldt for making it all happen.

Contents

Chapter 1

Introduction

1.1 Motivation

Designers and engineers are often confronted with problems that involve a large number of parameters, contradicting objectives, and a set of various constraints, resulting in a complex solution space. Generally, a top-down approach is applied and the problem is simplified by initially developing an abstract concept and then gradually refining it until a solution is reached. This process is guided by experience and intuition, and is regarded as a core trait of respected designers, architects and engineers. It has enabled and contributed to countless solved design problems and realized projects. However, this approach also leaves potential unexplored, as the designer is subconsciously biased towards a set of possible solutions originating from the same initial intention, where the extent of the true solution space remains largely unknown. Some design problems are simply too complex to be fully explored, making this top-down approach a necessity. In this thesis, a concept is presented that at its core challenges this assumption, by striving to augment the designer's exploration capabilities and reversing the traditional design process.

1.2 Proposed Solution

The proposed solution is limited to input-output design processes, as it is dependent on a dataset representative of the design instance and its performance. The classic example is a parametric design process.Traditionally, it follows the pattern of defining input parameters, modeling a design instance and evaluating its performance, based on a set of pre-defined performance attributes. Examples for these attributes include structural properties, used material, total cost, or any other attribute that is quantifiable and relevant to the design task. In the case of conditional design problems, where specific requirements need to be fulfilled, the traditional process demands the designer to iterate over the described procedure and continuously adjust the input parameters until a respectable performance is achieved. Not only is this time-consuming and cumbersome, it is also likely that potential solutions remain unexplored.

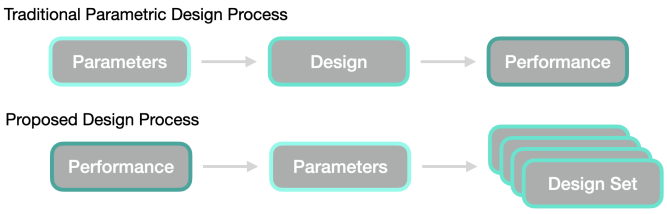


Figure 1.1: Traditional and proposed parametric design processes.

A deep learning concept is proposed that enables to reverse this process by learning the underlying relationship of input parameters and performance, empowering the user to generate a set of design instances fulfilling the required attributes and properties (figure 1.1). This bottom-up approach reduces the designers bias and allows for an efficient and more inclusive solution space exploration. The deep learning concept employed in this thesis is called *Conditional Variational Autoencoder*. It operates by compressing the input parameters into a regularized and low-dimensional latent space, encoding the inputs most important patterns and properties. This process is carried out by a network called the *encoder*. Conversely, a neural network called the *decoder* maps this latent space vector back to its original form, reconstructing its input. The model learns by comparing the original input with its reconstruction, adjusting the own parameters accordingly. Once the model is trained, it can *sample* from the probabilistic latent space and forward it through the generative model, the decoder. The output of this process is a novel design instance. At this point, the user has no control over this process, as the sampling and the generative model do not require any external inputs. By appending a conditionality to the latent vector, which usually is equivalent to the pre-defined performance attributes, the model additionally develops an understanding of the interconnected relation of the input and its respective performance. This imposed conditionality can be utilized to control the properties of the generated design. The user specifies the desired attributes and the model samples any number of designs from its latent space, meeting those requirements. For any given conditionality, the latent space fundamentally *is* the representation of the solution space, that is being interpreted by the decoder to produce diverse solutions once it has been substantiated (or sampled). Additionally, the model is encouraged to develop an analysis pipeline, enabling it to estimate the performance attributes of a given design instance. This allows the model to close the loop and evaluate its own performance of generating conditioned design instances.

1.3 Application

In this thesis, the CVAE concept is applied on the subject of acoustic panel generation. Acoustic panels absorb the energy of soundwaves in a thermodynamic transfer. They are used in concert halls, churches, restaurants, studios or in acoustic laboratories and have different requirements for each of those settings. We are provided with data representations of acoustic panels and the corresponding performance attributes. Based on this dataset, the CVAE model should ultimately be able to generate novel designs subject to a set of performance attributes and evaluate its own performance. As a guidance in the development process, a hypothesis is posed that states the expected requirements for the model to reach the objective.

- **Hypothesis** - Requirements for a functional acoustic panel generating model, conditioned on performance attributes:
    1. The model is capable of compressing a given acoustic panel and reconstruct it accurately.
    2. The model is able to accurately estimate the performance attributes of a given acoustic panel.
    3. The model maintains a regularized latent space.

Chapter 2

Related Work

The field of generative models is dynamic and rapidly evolving. Especially variational autoencoders (VAE) and generative adversarial network's (GAN's) have become popular concepts that are utilized for bayesian inference [12], image generation [2] or language processing [14] to name a few. Radford et. al. [11] provide a conclusive overview of the potential of GAN's in the field of supervised and unsupervised learning, whereas Oh et. al. [10] apply GAN's for design optimization and exploration. The theoretical ground-work of variational autoencoders is provided by Kingma et. al. [7] and Doersch et. al. [4], among others. Vahdat et. al. [13] propose advances in the architecture of the classical VAE in order to generate high-quality images and close the gap to GAN's, that are currently outperforming VAE's in this field. Most of the work that has been done on generative models is performed on open-source datasets and there seems to be untapped potential of actively employing these concepts on practical applications.

The main foundational building block for this thesis is provided by Salamanca et. al. with the work of *Augmented Intelligence for Architectural Design with Conditional Autoencoders: Semiramis case study* [9], where a machine learning model is developed that enhance architect's design experience. More specifically, a conditional autoencoder is proposed that reverses the parametric modeling process and allows the architect to define desired properties and obtain multiple design instances that fulfill them. As some of the proposed solutions have not been thought of before, it is an augmentation of human's understanding of the design task and aims at stimulating design exploration. The concept was applied on a vertical garden architectural project that was carried out in Zug, Switzerland in 2022. This thesis is a continuation of that work, with advancements in the models architecture by introducing variationality and a probabilistic latent space, generalizing the proposed approach. Additionally, a convolutional neural network is embedded, allowing the processing of images and alleviating the dependency on a classical parametric input structure, opening up a new dimension of potential applications. One of the main objectives is to validate the concept on a new and fundamentally different use case. Realizing a functional conditional variational autoencoder in the domain of acoustic panel generation would further solidify the already achieved accomplishments with the proposed concept.

Chapter 3

Methods

This thesis is conducted in a *Renku limited* environment with version 0.12.16. Renku describes itself as a platform that bundles together various tools for reproducible and collaborative data analysis projects. It is developed and maintained by the SDSC ETH. All computations have been carried out with the following system requirements:

- Python 3.7.6
- CPU: Intel Xeon Gold 6130 CPU @ 2.10GHz
- GPU: Tesla P100-PCIE-12GB
- RAM: 32GB

The code is based on a *pytorch lightning* infrastructure. There are three main components to it, the *data module*, the *model* and the *lightning trainer*. This structured approach allows a streamlined process and a simple implementation of wrappers, callbacks, and plug-ins. In this thesis, we make extensive use of the *Weights and Biases* logger, which allows to document and visualize the model evolution in the Weights and Biases dashboard. The high level structure of the code is shown in listing 3.1.

Listing 3.1: High-level code structure.

```
import pytorch_lightning as pl
from pytorch_lightning.loggers import WandbLogger
from classes import DataModel, Model

# define config with default parameters
config = default()

# load and preprocess data
data_module = DataModel(config)

# build cvae model
cvae_model = Model(config)

# initialize wandb logger
wandb_logger = WandbLogger()

# initialize trainer with data_module, cvae_model and wandb_logger
trainer = pl.Trainer(
                model=cvae_model,
                datamodule=data_module,
                logger=wandb_logger
                )

# fit model to data and visualize performance in wandb dashboard
trainer.fit()
```

3.1 Dataset

The dataset consists of eighty-three *.pkl* files, each taking up 50KB of storage in its original form. Combined they contain approximately 40'000 samples. A sample includes an image array of a sound panel, as well as thirty-one attributes that correspond to the given panel, packed in a pandas dataframe.

3.1.1 Performance Attributes

The performance attributes quantify the performance or fitness of a sound panel given the problem statement of the design assignment. Quantifiability and relevancy to the task at hand are requirements in order to be regarded as a performance attribute. In this dataset, thirty-one attributes and therefore thirty-one potential performance attributes are provided. The performance attributes are defined by the designer, by taking into account what the most critical aspects of the design problem are and how well the given attributes reflect them. Exploring the aspects of the task that are not as approachable to the designer and filling that void with the performance attributes is at the core of true augmentation. Ultimately, the objective of this exercise is to empower the designer's abilities in solving the design task.
In this thesis, the final set of performance attributes should enable the designer to control the panel generation in a multitude of ways with a manageable number of parameters. Obviously, the actual performance needs to be captured, but it is just as valuable to represent certain characteristics that do not translate to classical quantifiable metrics, but rather to subjective features such as shape or texture. Having a high-dimensional solution space is where the designer initially started, so that is to be avoided. For this thesis, the performance attributes are selected as follows:

- fft mean x
- fft mean y
- fft mean
- panel height
- high res x
- high res y

The *fft* parameters are regarded as the most important attributes, since the purpose of a sound panel is to modify sound waves, which is described by the *fft* values. The *panel height* is a very crucial parameter in its own right, as the height dimension often is a hard constraint and important to have control over. The two *high res* attributes describe geometrical characteristics of the panel in the x and y directions. They enable the designer to enforce structural preferences on the panel generation. If this set of performance attributes can be learned well by the model, it enables the designer to impose diverse constraints on the generational model and guide the sound panel generation process accordingly. The performance attributes are referred to as $\boldsymbol{x}$.

3.1.2 Panels

The panels are packed in the form of a numpy array with an initial dimension of

$$init\_dim = (600, 600). \tag{3.1}$$

They are single-channel or greyscale images with datatype $uint8$, a discrete integer value range of $[0, 255]$. The pixel value is equivalent to the z coordinate of the sound panel at the given location, encoding the topology of the panel. These types of 2d images are also referred to as *heatmaps*. In order to get the images compatible with a deep learning network, the data needs to be transformed into a continuous datatype. Since $uint8$ images are taking up only a fraction of the memory compared to i.e. $float64$, it is important to be conscious of memory usage going forward.

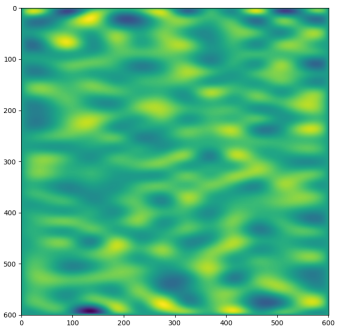


Figure 3.1: Original panel with 600 x 600 pixels.

The panels contain 360′000 datapoints each, which leads to large data batches. Therefore, the panels are transformed in order to make them more manageable. The x and y axis are evenly split into three sections, creating a grid of nine *sub-images*. From this new set of *patched* images, two of the sub-images are randomly selected and added to a new dataset. There is a specific reason why it is not trivial to add any arbitrary number of sub-panels to the new dataset, apart from memory concerns. The sub-panels share the same set of performance attributes, which are expected to be of continuous nature. Having multiple of instances in a dataset with exactly the same values goes against the expectation of continuous scales and might negatively impact the behaviour of the deep learning model. On the contrary, it is beneficial to increase the dataset as long as the data is meaningful and representative. Considering these factors, down-sampling to two sub-images from each original panel is reasonable and effectively doubles the number of samples in the dataset.

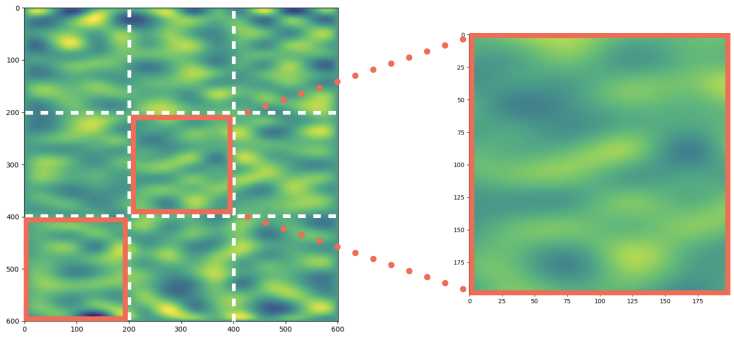


Figure 3.2: Random sampling of sub-panels from original panel.

There are some assumptions made and uncertainties taken into account during this process. It is not clear that the sub-panels still have the same impact on sound waves as their performance attributes would suggest, leaving open the possibility of having an inaccurate dataset. Since this inaccuracy is likely to be insignificant and down-sampling the images to some degree is necessary due to memory usage, it is justifiable to regard the performance attributes of the down-sampled panels as valid. There is one attribute where this small inaccuracy can be resolved, as the *panel height* parameter corresponds to the largest pixel value in the image array. Therefore, the new panel height performance attribute is set as

$$panel\_height_{new} = max(sub\_panel). \tag{3.2}$$

In a mathematical context, the sub-panels are referred to as w from here on. Figure 3.3 visualizes the translation of w to a 3d environment and helps to build an intuition regarding the 2d representations of the sound panels.

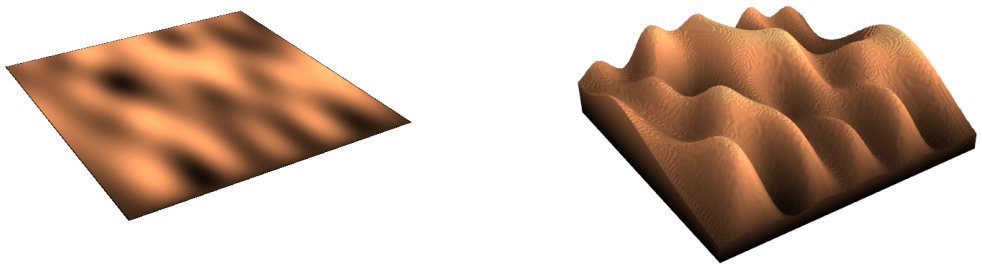


Figure 3.3: 2d sound panel (left) and 3d model of sound panel (right).

3.1.3 Preprocessing and Analysis

In order to stabilize and accelerate the computational procedure of the deep learning model, the data is transformed and normalized as shown in table 3.1.

variable	normalized range	initial dtype	new dtype
w	$[-1, 1]$	$uint8$	$float32$
x	$[0, 1]$	$float64$	$float32$

Table 3.1: Data preprocessing.

Once the data is preprocessed, it is split into a *train*, *validation* and *test* subset. The validation set is examining whether the model is overfitting the training data and the test set is used for a final evaluation, testing if the hyperparameter tuning is leading to an overfit of the validation set. The data is split relative to the full dataset size as detailed in table 3.2.

subset	relative share
train	0.8
validation	0.1
test	0.1

Table 3.2: Dataset split.

After transforming the panels and defining the performance attributes, the distribution of the performance attributes x within the dataset is investigated. Additionally, it is important to build an understanding of how x correlates with the topology of w. In figure 3.4, a histogram of the fft mean and high res performance attributes is plotted relative to their respective normalized value. The full set of histogram for all performance attributes is displayed in the appendix in figure A.13. The dashed, red lines indicate the 0.1 respectively 0.9 values of its cumulative distribution. This gives some additional intuition in understanding the histogram plot of x and its distribution in the dataset.

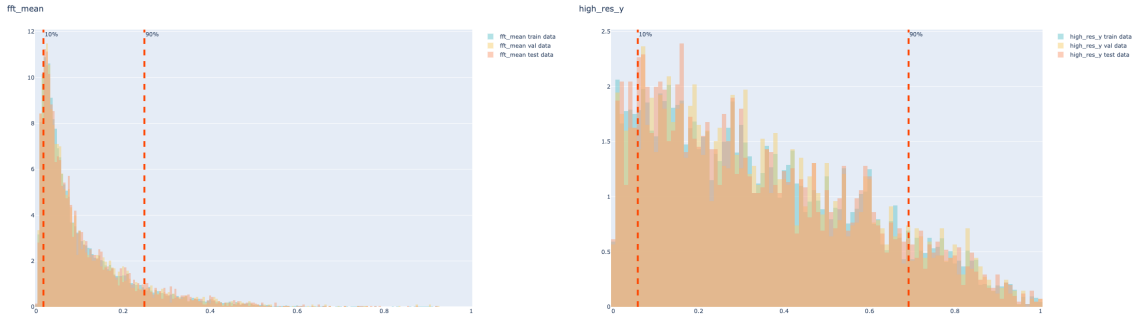


Figure 3.4: Distribution of the normalized performance attributes.

When examining the plots, the distribution of the *fft mean* attribute is resembling a *gamma* distribution, heavily shifted towards its zero point. The 0.9 border lines of the respective cumulative distribution $\mathcal{F}$ is well below $x = 0.3$, which also can be expressed as $\mathcal{F}_{fft\_mean}(0.3) > 0.9$. The *fft mean x* and *fft mean y* attributes are distributed similarly. This means that for a large part of the *fft* attribute ranges, very little data is available. The *high res y* attribute, is linearly distributed without any significant outliers. The full $\mathbf{x}$ data range $[0, 1]$ is represented fairly well. The distribution of the *high res x* attribute has the same properties except for a flipped slope, as displayed in figure A.13. This can be explained by the fact that *high res y* is always larger than *high res x*, or

$$x_{high\_res\_x,i} < x_{high\_res\_y,i} \forall i. \tag{3.3}$$

As a result of equation (3.3) the panels are always oriented in the same direction. The *panel height* histogram plot shown in the appendix suggests that the panels are likely generated from a discrete set of values, as implied by the frequent spikes in the distribution. A large portion of the panels are at maximum height, considering the large spike at $x = 1$. Fortunately, the gaps between the various spikes are filled to some degree due to the reassignment of the new *panel height* detailed in equation (3.2), since not all sub-panels share the same height the original panel was generated with. This introduces some continuity into the distribution and dampens its discrete character.

(a) fft mean

(b) high res y

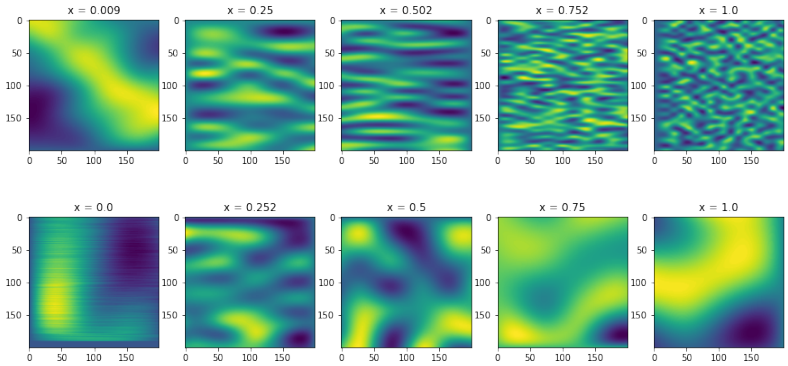


Figure 3.5: Panel topography relative to the performance attribute distribution.

Figure 3.5 demonstrates the panels topology transition relative to the performance attribute distribution. A selection of panels is plotted that correspond to the evenly spaced values throughout the distribution of a given performance attribute. For consistency, the panels of the attributes *fft mean* and *high res y* are examined. The full set of figures is shown in figure A.14. The *fft mean* attribute has an easily identifiable effect on the panels topology, resulting in fragmented panels when

approaching the $x = 1$ boundary. As for the *high res y* attribute, the geometrical characteristics that are imposed on the panels are best examined by comparing the two boundaries. Apparently the *high res y* attribute regulates the gradients of an image array in the y direction. As evident in the appendix figure A.14, the same properties that are observed in figure 3.5, are translated to *fft mean x*, *fft mean y* and *high res x* as expected. There are no takeaways from the *panel height* images, since the color mapping of each plot is scaled accordingly to its relative value range. In figure 3.6, the *fft* and *high res* edge cases are translated to a 3d environment. It is evident that the large gradients of these sound panels are not desirable and will be regarded as infeasible.

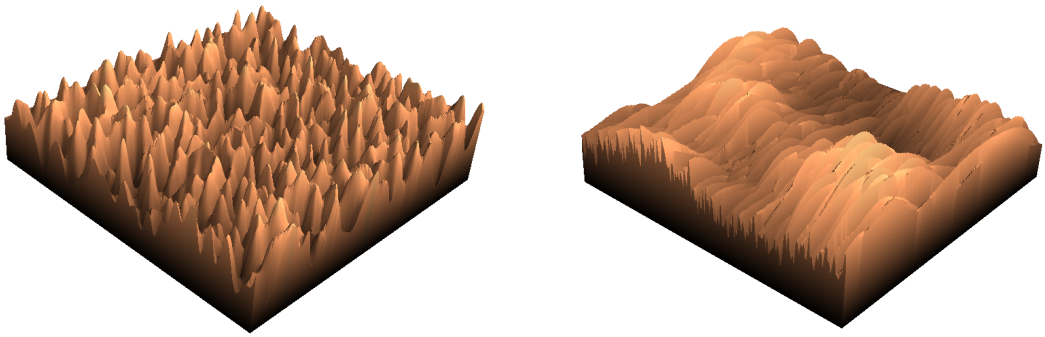


Figure 3.6: Example of potentially infeasible sound panels

In order to address the issues that have been raised, a clamping of the dataset is introduced. There are two main problems that are being confronted with the clamping:

- removing outliers
- removing infeasible sound panels

As detailed in table 3.3, all samples with *fft mean* attribute values above 90% of the corresponding distribution are clamped, as well as all samples with *high res y* attribute values below 10% of the distribution. After the clamping, the performance attributes are being re-normalized.

performance attr	lower bound	upper bound
fft mean x	0	0.9
fft mean y	0	0.9
fft mean	0	0.9
panel height	0	1
high res x	0.1	1
high res y	0.1	1

Table 3.3: Clamping of dataset.

There are negative effects of this clamping procedure, with the first one being that the size of the dataset is reduced by approximately 25%. Secondly, the model needs to be more precise in order to reach the same level of relative performance as the normalization has spread out panels with attributes that used to be regarded as similar. This effect is only due to the scaling of the relative errors and has no actual impact on performance, but it is noteworthy that the model is likely to have reduced losses on the original dataset, assuming normalization.

3.1.4 Final Dataset

The final dataset contains a total of 61′050 samples. In order to validate the procedures from the previous section, the performance attribute density and the corresponding sound panel plots are compared with the ones from figure 3.4. As can be seen in figure 3.7, the plots show a better data representation over the full $[0, 1]$ range for the *fft mean* values and the extreme outliers are no longer present. The complete set of histogram plots is documented in figure A.15 of the appendix.

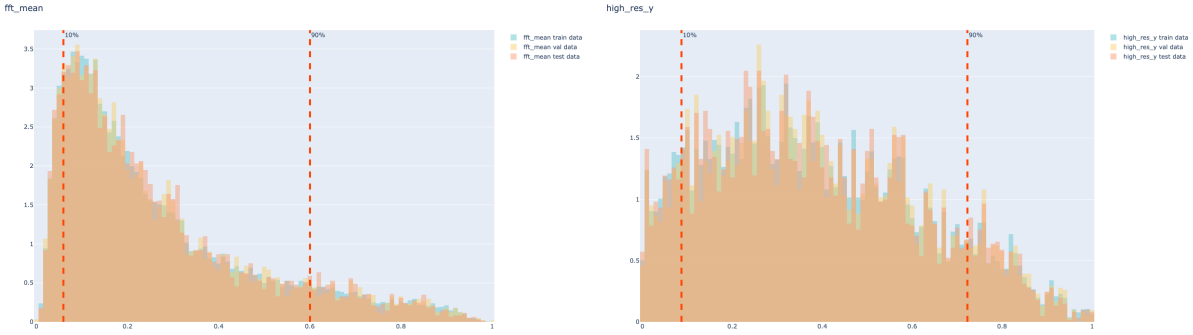


Figure 3.7: Clamped histogram plots of fft mean and high res y attributes.

For the second objective of the clamping procedure, the sound panel topology transitions shown in figure 3.8 are inspected. The edge case panels are not as extremely fragmented as before the clamping for the *fft mean* panels and the *high res y* seem more manageable as well. Although they still show some of the same characteristic, these sound panels are assumed to be feasible. The complete set of topology transitions is documented in figure A.16 of the appendix. This last validation concludes the dataset evaluation and the focus shifts toward the neural network model.

(a) fft mean

(b) high res y

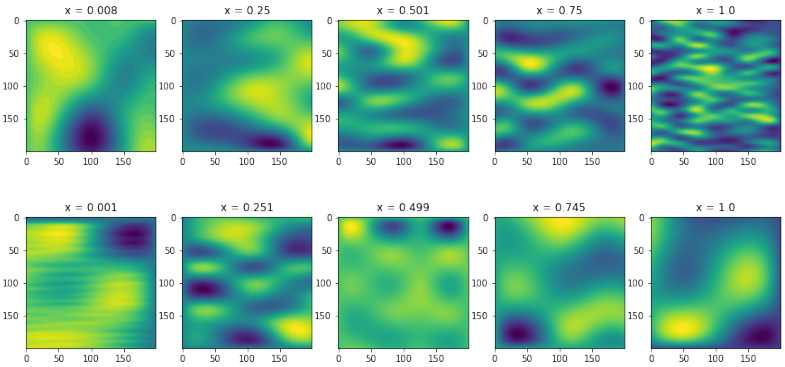


Figure 3.8: Panel topography plots of clamped fft mean and high res y attributes.

3.2 Conditional Variational Autoencoder

A *Conditional Variational Autoencoder* is based on the concept of an autoencoder, as shown in figure 3.9. A classic autoencoder consists of two neural networks, an encoder $\boldsymbol{q(.)}$ and a decoder $\boldsymbol{p(.)}$. The encoder compresses the input into the latent space vector $\boldsymbol{z}$, or $\boldsymbol{z = q(input)}$. Generally, the input is high-dimensional and the latent vector $\boldsymbol{z}$ low-dimensional, but $\boldsymbol{z}$ should still be representative of the input. The latent space $\boldsymbol{z}$ is also referred to as the *bottleneck*. The decoder takes $\boldsymbol{z}$ as input and expands it back to its original size, $\boldsymbol{output = p(q(input))}$. The performance is typically measured by computing the reconstruction error $\boldsymbol{|input - output|}$.

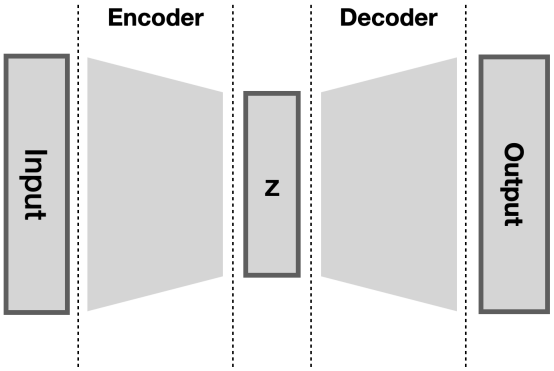


Figure 3.9: Autoencoder.

There are limitations to the classical autoencoder concept. It is not trivial to use it as a generative model, since its latent space is unregulated. This can lead to *blindspots* within the latent space that produce strange outputs with bad performance. Additionally, it is not possible to *request* a specific type of generated output, since there is no conditionality included. It is predominantly used in settings where it does not need to generate a new design or object. Classical use-cases include de-noising or imputing images, among others [1]. On the other hand, the variational autoencoder model is specifically designed to take on generative design tasks. It deals with the autoencoder's shortcomings by introducing a regulated, probabilistic latent space. This produces a smooth transition between slight adjustments in the latent space and generated designs, eliminating the blind spots. However, the probabilistic nature of the model introduces new challenges as well. Stochastic nodes prohibit gradient computation and therefore backpropagation, which is a the core process of a neural network such as the autoencoder. In order to enable backpropagation, a new method called the ***reparametrization trick*** is employed [7]. The idea is to express the probabilistic variable $\boldsymbol{z}$ as a deterministic variable. This is achieved by introducing an auxiliary variable $\boldsymbol{\epsilon}$, allowing to redefine the probabilistic latent space variable $\boldsymbol{z}$ as

$$\begin{aligned} z &= \mu + \sigma * \epsilon, \\ \epsilon &\sim \mathcal{N}(0, 1). \end{aligned} \tag{3.4}$$

The mean $\boldsymbol{\mu}$ and the standard deviation $\boldsymbol{\sigma}$ are neurons in the final layer of the encoder. With the reparametrization trick, $\boldsymbol{z}$ has successfully been reformulated as a differentiable variable while maintaining its probabilistic character. $\boldsymbol{\epsilon}$ allows the generation of any arbitrary number of novel designs, by sampling the latent space and passing it to the generative model, the decoder.

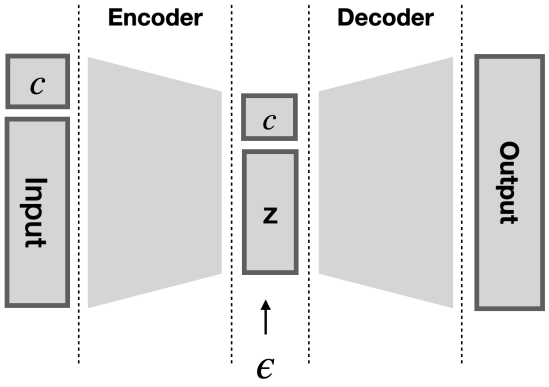


Figure 3.10: Conditional Variational Autoencoder.

In order to introduce *conditionality* to the model, the vector $\boldsymbol{c}$ is simply being appended to the latent space $\boldsymbol{z}$. This small modification enables the decoder to generate designs that are conditioned on $\boldsymbol{c}$, which is one of the core objectives of this thesis. The model is no longer encouraged to encode information of $\boldsymbol{c}$ and as a consequence the latent space is detaching from $\boldsymbol{c}$ specific characteristics, enabling it to focus on the solution variability within the space of $\boldsymbol{c}$. To enhance this welcomed effect, the $\boldsymbol{c}$ vector is additionally appended to the input as shown in figure 3.10.

Objective Function

The objective function of the conditional variational autoencoder is given in [12] as:

$$\log p(w|x) - D_{KL}[q(z|w,x)||p(z|w,x)] = E[\log p(w|z,x)] - D_{KL}[q(z|w,x)||p(z)], \tag{3.5}$$

where $\boldsymbol{p(w|x)}$ is the dataset, D_{KL} is the KL-divergence, $\boldsymbol{q(z|w,x)}$ is the approximated posterior, $\boldsymbol{p(z|w,x)}$ is the true posterior, $\boldsymbol{p(w|z,x)}$ is the likelihood and $\boldsymbol{p(z)}$ is the prior. As evident in equation (3.5), all of the terms are conditioned on the performance attributes $\boldsymbol{x}$, except for the prior. The true posterior $\boldsymbol{p(z|w,x)}$ is generally intractable, so the objective function needs to be reformulated before it can be applied in a real-world scenario.

CVAE Loss

This is where the CVAE loss or *evidence lower bound* (ELBO) comes into play. It is a reformulation of equation (3.5) that eliminates the true posterior from the equation and can directly be expressed as a loss function:

$$ELBO = min(E_q[\log q(z|w,x) - \log p(z)] - E_q \log p(w|z,x)), \tag{3.6}$$

where ELBO is to be minimized. The first term of the RHS is the **KL loss** and the second term is simply the log likelihood or in more general terms the **reconstruction loss**. The variables can be expressed in terms of the CVAE model in figure 3.10, where $\boldsymbol{q(z|w,x)}$ and $\boldsymbol{p(w|z,x)}$ represent the encoder and decoder respectively. Initially, $\boldsymbol{z}$ is sampled from $\boldsymbol{q(z|w,x)}$, the encoder. Subsequently, the KL divergence of $\boldsymbol{q(z|w,x)}$ and the prior $\boldsymbol{p(z)}$ is computed, as well as the log likelihood or $\log \boldsymbol{p(w|z,x)}$, which is given by the generative model, the decoder. The log likelihood is analogous to the density of data point $\boldsymbol{w_i}$ under the generative model, given $\boldsymbol{z_i}$ and $\boldsymbol{x_i}$. The prior can take the form of any desired distribution and acts as the **regularization** term in the VAE loss 3.6. In this thesis, it is modeled as

$$p(z) \sim \mathcal{N}(0,1), \tag{3.7}$$

therefore encouraging $\boldsymbol{q(z|w,x)}$ to be distributed as a multivariate standard Gaussian. Since the KL divergence is carried out on an element basis and computed for each input $\boldsymbol{w_i}$ separately, it is

not desirable to fully minimize the KL loss. A KL loss of zero is equivalent to an encoder $\boldsymbol{q(z|w,x)}$ that is independent of $\boldsymbol{w}$ and $\boldsymbol{x}$, being identical to the prior and producing fully generalized samples. On the contrary, the model will behave similarly to a regular Autoencoder if no KL loss is enforced. The standard deviation parameters of the latent space distribution $\boldsymbol{q(z|w,x)}$ for a given input $\boldsymbol{w}$ will be vanishingly small, effectively eliminating the encoder's probabilistic nature.

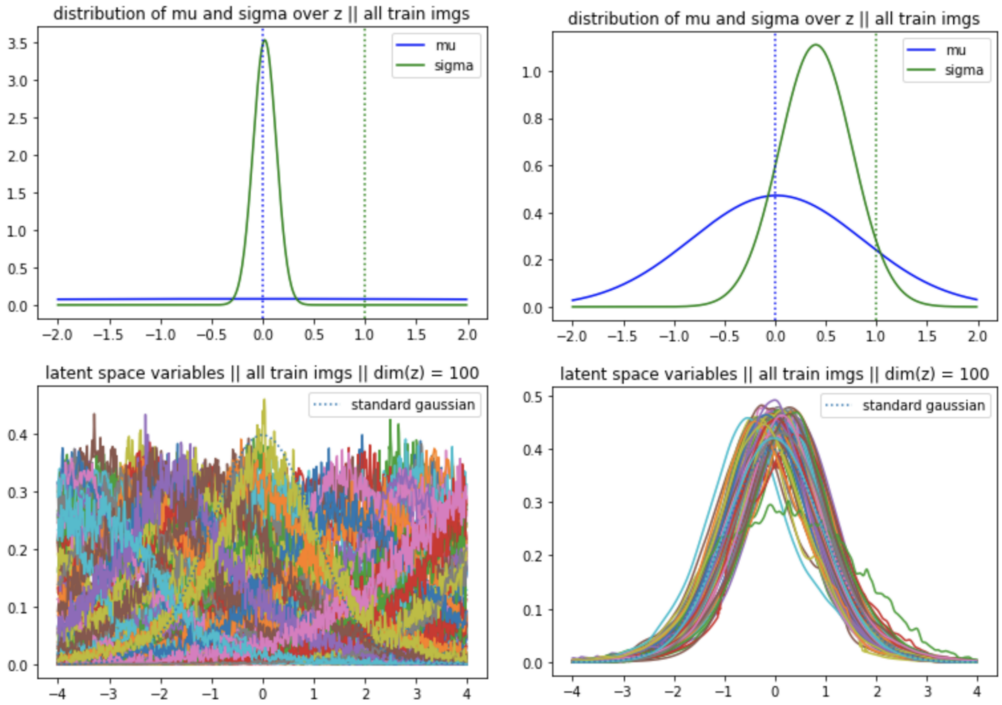


Figure 3.11: Visualized latent space with KL divergence loss ignored (left) and enforced (right).

Figure 3.11 demonstrates this phenomenon, where the upper plots display the distribution of the $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ parameters of $\boldsymbol{q(z|w,x)}$ and the lower plots show the approximated resulting latent space distribution for each $\boldsymbol{z_j}$ separately. On the two left plots, it can be observed that the standard deviation parameter $\boldsymbol{\sigma}$ for a single sample are virtually zero, while $\boldsymbol{\mu}$ is unconstrained. On the right plots, it is evident that $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are regularized by the prior, resulting in a latent space distribution resembling the standard Gaussian. The approximated latent space distribution is produced by forwarding all train samples through the model and registering the distribution of each $\boldsymbol{z_j}$. These distributions are averaged out, resulting in an estimate of the true latent space of the model as described in equation (3.8).

$$q(z_j|w,x)_{true} \simeq \frac{1}{n}\sum_{i=0}^{n-1} q(z_j|w_i,x_i), \quad \forall j. \tag{3.8}$$

3.3 Model Architecture

In this thesis, the CVAE model introduced in section 3.2 is employed to take on the challenge of generating sound panels conditioned on performance attributes. As the sound panels are described in the form of 2D gray-scale images, the model architecture consists of *Convolutional Neural Networks's* (cnn) in combination with fully-connected *Feed-Forward Network's* (ffn) to encode and decode the panels and attributes. A forward pass through the encoder and decoder is described as follows:

1. **Encoder - $q(z|w,x)$**
   (a) $wz \leftarrow q_{cnn}(wz|\boldsymbol{w})$
   (b) $\mu_z, \sigma_z \leftarrow q_{ffn}(z|wz, \boldsymbol{x})$
   (c) $\boldsymbol{z} = \mu_z + \epsilon * \sigma_z$, where $\epsilon \leftarrow \mathcal{N}(0,1))$
2. **Decoder - $p(\hat{w}|z,x)$**
   (a) $zw \leftarrow p_{ffn}(zw|\boldsymbol{z}, \boldsymbol{x})$
   (b) $\hat{\boldsymbol{w}} \leftarrow p_{cnn}(\hat{w}|zw)$

This network is visualized in figure 3.12.

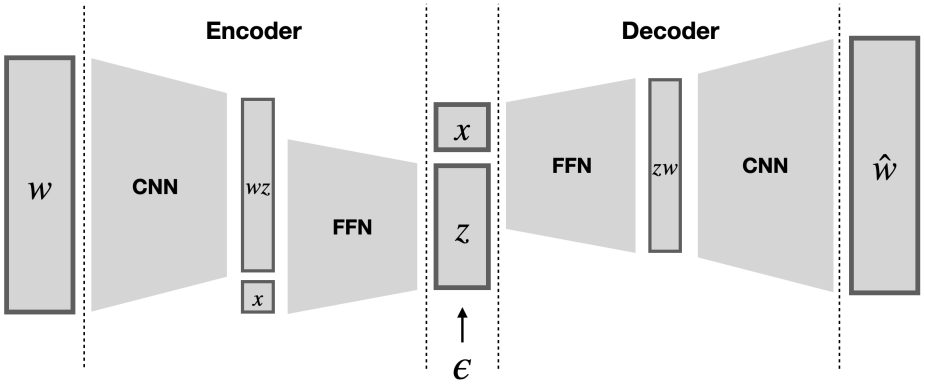


Figure 3.12: CVAE detailed architecture.

3.3.1 Attribute Reconstruction

In order to evaluate the performance of the CVAE model in generating sound panels conditioned on a set of performance attributes, a measuring tool is needed. Unfortunately, we do not have access to the analysis process that computes the *true* performance attributes given a 2D mapping of a sound panel. Therefore, a process that approximates the true attributes given an image is needed, which is defined as $\boldsymbol{q(\hat{x}|w)}$. This process is called *attribute reconstruction*. Naturally, a neural network is employed to take on this task. In an effort to reduce computational costs and memory usage, this new process is facilitated within the encoder of the already existing CVAE model. Whereas the cnn of the encoder is reused, a new feed-forward network is introduced. The two ffn's within the encoder are differentiated by adding a prefix to their name which relates to their output, **zffn** and **xffn**. The new network is visualized in figure 3.13 and the corresponding attribute reconstruction process is described as:

- **Encoder** - Attribute Reconstruction - $\boldsymbol{q_{attr}(\hat{x}|w)}$
  1. $wz \leftarrow q_{cnn}(wz|\boldsymbol{w})$
  2. $\hat{\boldsymbol{x}} \leftarrow q_{xffn}(\hat{x}|wz)$

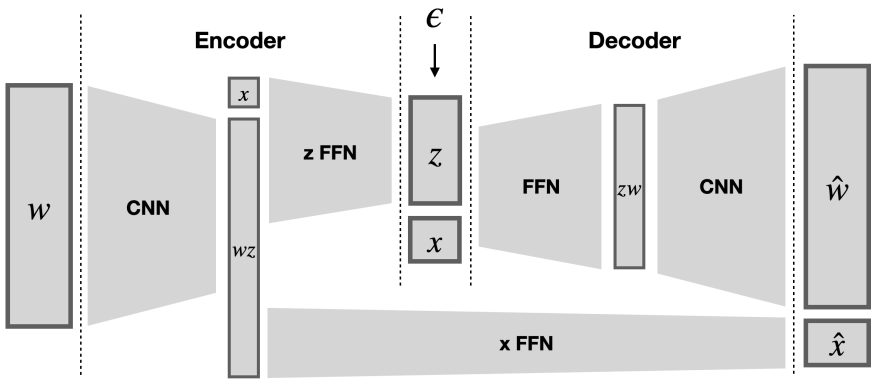


Figure 3.13: CVAE model incorporating the attribute reconstruction process.

The approximation of $\boldsymbol{x}$ enables an evaluation of the CVAE model's performance in its most important task: generating conditioned sound panels. However, the quality of this evaluation is directly dependent on the accuracy of $\hat{\boldsymbol{x}}$ and the performance of the attribute reconstruction process. This introduces a new level of uncertainty into the thesis, which must be minimized by all means possible.

3.3.2 Neural Networks

In this section, the high-level model presented in section 3.3 is defined in more detail. It is not obvious what kind of network design will perform best given the design problem. Since there are a large number of parameters available that define a neural network, a structured approach is needed so that the final number of possible parameter combinations is manageable. The first simplification to be introduced is the concept of the **cnn** and **ffn blocks**. They describe a recurring set of layers that can be reduced to a single block unit.

CNN Block

The *cnn block* is responsible for creating a meaningful and compressed representation of input $\boldsymbol{w}$, by filtering for patterns and other image features, as well as reversing this whole procedure in the case of the decoder. The layers that compose a cnn block are:

- Convolutional Layer (conv),
- Transposed Convolutional Layer (conv tr),
- Maxpool Layer (maxpool),
- ReLU Layer (relu),
- Batch Norm Layer (norm),

where the encoder employs regular and the decoder transposed convolutional layers. In the encoder, the maxpool layer is perfectly suited to compress the image to the desired shape. Upsampling the image is not as simple, as a transposed maxpool layer leads to a bad reconstruction performance. Therefore, transposed convolutional layers are used to expand the image by setting the *stride* equal to the *kernel-size* of the maxpool layer. However, when applying $stride > 1$ on a transposed convolutional layer, the resulting output dimension is underdefined. Since the output image $\hat{\boldsymbol{w}}$ is required to have identical dimensions as the input $\boldsymbol{w}$, more information is needed. This information is provided in the form of the *output padding*, which fully defines the output dimension of a transposed convolution. With the methodology of compressing and expanding images within a cnn block unit in place, three separate versions of cnn blocks are introduced in table 3.4.

cnn block label	coder	set1	set2	set3
21	encoder	conv	maxpool + relu	
	decoder	conv tr + relu		
22	encoder	conv	maxpool + relu	
	decoder	conv tr + relu	conv tr + relu	
32	encoder	conv + relu	conv	maxpool + relu
	decoder	conv tr + relu	conv tr + relu	

Table 3.4: cnn block definition.

The cnn block label indicates how many sets of layers it contains, where the first digit represents the encoder and the second the decoder block. In order to fully define the cnn block, the *kernel-size* of the convolutional and maxpool layer need to be provided. The stride and padding are always set to one.

FFN Block

The *ffn block* is the intersection of the cnn neural network and the latent space. It compresses and maps the patterns and features from $\boldsymbol{wz}$ to $\boldsymbol{\mu_z}$ and $\boldsymbol{\sigma_z}$, from which $\boldsymbol{z}$ is sampled, and vice versa. Due to its one-dimensional nature, defining the input and output size is trivial. The ffn blocks are composed from the following layers:

- Linear Layer (linear),
- ReLU Layer (relu),
- Batch Norm Layer (norm),
- Dropout Layer (dropout),.

The dropout layer is a tool to decrease overfitting by zeroing out the weights and biases of neurons with a probability p_{drop}. The ffn block layer sequence is defined in table 3.5.

	set1	set2
ffn block	linear + relu	dropout

Table 3.5: ffn block definition.

The number of neurons that are present in the linear layer is defined by the parameter *ffn layer size*. The probability p_{drop} is further defined in a later section. Additionally, for the five sub-networks of the model, there is the option of adding a batch norm layer at the end of all contained cnn and ffn blocks. This introduces the following five new parameters into the system:

- enc cnn norm
- enc zffn norm
- enc xffn norm
- dec ffn norm
- dec cnn norm

Since *enc cnn norm* and *dec ffn norm* are at the beginning of their respective networks, they are automatically set to *True*. By default, all batch norm and relu layers are disabled for the final cnn or ffn block of each sub-network, to prevent the transformation of a final output.

Network Design

The network design employs the blocks defined in this section and arranges them in different depths and constellations. Three network designs are introduced in table 3.6.

net design	n cnn blocks	conv kernel	maxpool kernel	n ffn blocks	ffn layer dim
regular	4	6	2	3	512
regular ffn	4	6	2	3	1024
deep	5	3	2	4	1024

Table 3.6: Net design variations.

Due to memory usage, the dimensions of the two-dimensional **wz** and **zw** tensors should be similar for all three network designs. The difference of **wz**'s size being $(10, 10)$ or $(20, 20)$ is a 400% increase in pixels, which directly correlates to the number of input/output layer neurons of **ffn** networks. Since the size of **wz** is mostly dependent on the number of cnn blocks and the corresponding conv and maxpool kernel-sizes, these parameters need to balance each other out. The three designs presented in table 3.6 all produce a similar size of **wz**, since the *deep* network has a smaller *conv kernel* of size three. Combining the three net design variations with the three cnn block variations gives nine possible designs for the neural network. The latent space dimension is still a free parameter. As for the *channels*, they are defined identically for all design variations. In the first layer there is a single channel, in the second eight, and from there the channels are doubled with each cnn block to follow, as given in equation (3.9).

$$\begin{aligned} &channels_0 = 1, \\ &channels_i = 8 \cdot 2^{i-1}, \quad i \in [1, n_{cnn\_blocks} - 1]. \end{aligned} \tag{3.9}$$

Table 3.7 summarizes the parameters that were introduced in this section.

free parameters	value	fixed parameters	value
cnn block			
conv kernel		stride	1
maxpool kernel		padding	1
ffn layer size			
enc xffn norm		enc cnn norm	True
enc zffn norm		dec ffn norm	True
dec cnn norm			
net design		channels	(1, 8, 16, 32, ...)
latent space dim			

Table 3.7: block parameter definitions.

3.4 Model Loss Function

The CVAE loss described in equation (3.5) is focused on the trade-off between the reconstruction of $\boldsymbol{w}$ and the KL divergence of encoder $\boldsymbol{q(z|w,x)}$ and prior $\boldsymbol{p(z)}$. The problem statement has been slightly adapted with the introduction of the attribute reconstruction. The CVAE's ability to generate panels is not the only concern, given the questionable quality of its performance evaluation. Therefore, the model needs to be encouraged to reconstruct attributes accurately, while developing and improving image compression and reconstruction. This needs to be articulated in the form of a multi-objective loss function, expanding on the already existing CVAE loss. Since it is not clear

how these different objectives are ideally prioritized, λ weight parameters are introduced. The generalized, multi-objective loss function is defined as:

$$loss_{tot} = \lambda_w loss_w + \lambda_x loss_x + \lambda_k loss_k \tag{3.10}$$

As formulated in the hypothesis, the expectation is that if the model is able to accurately reconstruct the panels $\boldsymbol{w}$ and the corresponding performance attributes $\boldsymbol{x}$, while maintaining a generalized latent space enforced by D_{KL}, all necessary tools are in place to generate panels conditioned on $\boldsymbol{x}$ and validate its own performance.

3.4.1 W Reconstruction Loss

$loss_w$ and λ_w are responsible for leveraging the reconstruction of the sound panel images. As detailed in equation (3.5), classically this loss is defined as

$$log\_likelihood = E_q \log p(\hat{w}|z, x)). \tag{3.11}$$

In general, the output of the decoder are parameters which define the distribution $\boldsymbol{p(\hat{w}|z, x)}$. As shown in equation (3.11), sampling $\boldsymbol{z}$ from the encoder $\boldsymbol{q}$ and ensuingly taking the expectation of the log distribution $\boldsymbol{p(\hat{w}|z, x)}$, yields the log-likelihood. As it is a negative value, it is subtracted from the total loss. In the case of single-channel images, like the sound panels from our dataset, the output of the decoder $\boldsymbol{p}$ can directly be interpreted as an image. This gives the option of replacing the log likelihood loss with the *mean-squared-error* (MSE) of $\hat{\boldsymbol{w}}$ and $\boldsymbol{w}$ which is defined as

$$w\_mse = \frac{1}{n} \sum_{i=0}^{n-1} mean((\boldsymbol{p(\hat{w}_i|z_i, x_i)} - w_i)^2), \tag{3.12}$$

where n is the batch size. We use the average of the MSE instead of its sum, since it increases the interpretability of $loss_w$ and makes it pixel- and batch-size agnostic. As of now, it is not clear which of the two losses will lead to a better performance of the model. Therefore, the parameter *log likelihood* is introduced that enables or disables the log likelihood loss. In case of a disabled log likelihood parameter, the contributions to $loss_w$ from images with flat topologies are less significant due to the nature of the MSE loss. In order to counter this behaviour, a scaled version of $loss_w$ can be introduced by enabling the parameter *rel loss*. The relative loss is defined as

$$loss_{w,rel} = \frac{loss_w}{\Delta + \epsilon}. \tag{3.13}$$

where Δ is defined as the pixel range of $\boldsymbol{w}$ and ϵ is a vanishingly small number for numerical stability.

Gradient Loss

In addition to the regular $\boldsymbol{w}$ reconstruction loss, we are experimenting with a gradient loss using the *Sobel* filter, as first introduced in [5]. In early test settings it became evident that the reconstruction of the panels was mostly suffering from the inability of producing sharp edges, which is a known problem in cvae models [13]. Since the Sobel filter is mainly used for *edge detection* practices, it seems to fit the objective perfectly. The Sobel filter is defined as

$$G_x = \begin{Bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & b & -1 \end{Bmatrix}, \quad G_y = \begin{Bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{Bmatrix}. \tag{3.14}$$

The $\boldsymbol{G_x}$ and $\boldsymbol{G_y}$ filters are imposed on the image by setting it as the kernel of a 2d convolutional layer $\mathbf{conv_{Sobel}}$. Therefore, the gradient loss $loss_g$ is defined as

$$loss_g = MSE(conv_{Sobel}(\hat{w}) - conv_{Sobel}(w)). \tag{3.15}$$

In order to incorporate the gradient loss into the models objective, the total loss defined in equation (3.10) is adjusted to

$$loss_{tot} = \lambda_w loss_w + \lambda_x loss_x + \lambda_k loss_k + \lambda_g loss_g. \tag{3.16}$$

3.4.2 X Reconstruction Loss

The reconstruction loss of x is relatively straight-forward. We will simply use the MSE loss in order to get a measure of how well the performance attributes have been reconstructed. Again, the loss is averaged over the performance attributes to enhance interpretability and make the loss attribute-size agnostic. It is defined as

$$loss_x = \frac{1}{m} \sum_{j=0}^{m-1} (\mathbf{q}(\hat{\mathbf{x}}_\mathbf{j}|\mathbf{w}) - x_j)^2, \tag{3.17}$$

where m is the total number of performance attributes.

3.4.3 KL Divergence Loss

The primary approach to applying the *kl divergence* is defined as

$$loss_k = \frac{1}{d_l} \frac{1}{n} \sum_{l=0}^{d_l-1} \sum_{i=0}^{n-1} E_q[\log \mathbf{q}(\mathbf{z_l}|\mathbf{w_i}, \mathbf{x_i}) - \log \mathbf{p}(\mathbf{z_l})], \tag{3.18}$$

where d_l is the dimension of the latent space and n is the batch-size. Unfortunately $\boldsymbol{q(z)_{true}}$ is not attainable, therefore the kl divergence is applied on an element basis to the easily accessible $\boldsymbol{q(z|w_i, x_i)}$, as defined in equation (3.18). Since $\boldsymbol{q(z|w_i, x_i)}$ is a bad approximation of $\boldsymbol{q(z)_{true}}$, the goal is not to *optimize* $\mathbf{q}(\mathbf{z}|\mathbf{w_i}, \mathbf{x_i})$ but to *regularize* it, assuming that the regularization will lead to a $\boldsymbol{q(z)_{true}}$ that resembles the prior. As is usually the case with regularization, too much over-generalizes the model while too little leaves it unregularized, as demonstrated earlier in figure 3.11.

Hereby a secondary approach to the kl divergence is introduced, that attempts to *optimize* the $loss_k$ instead of using it as a tool for regularization. In order to achieve this, a better approximation of $\boldsymbol{q(z)_{true}}$ is needed. Instead of sampling $\boldsymbol{z}$ from $\boldsymbol{q(z|w_i, x_i)}$ and computing the difference of the log likelihoods as described in equation (3.18),the *probability density function* (pdf) of $\boldsymbol{q(z)_{true}}$ is approximated numerically.

Listing 3.2: Secondary KL loss computation

```
def sec_kl_loss(batch_mean, batch_std):

# 1. define set of q given batch of mu and std parameters, as well as prior p
q_set = Normal(mean=batch_mean, std=batch_std)
p = Normal(mean=0, std=1)

# 2. compute q and p log probabilities of pre-defined x_values
x_values = linspace(start=-4, end=4, steps=batch_size)
q_set_log_prob = log_prob(q_set, x_values)
p_log_prob = log_prob(p, x_values)

# 3. get approximation of q true log probabilities by averaging over batch a
q_approx_log_prob = mean(q, dim=batch_dim)

# 4. take absolute difference of approximated q true and prior log probabilites
kl_div_sec = abs(q_approx_log_prob - p_log_prob)

# 5. average the secondary kl divergence loss over x values and latent space
return mean(kl_div_sec, dim=all)
```

The procedure described in listing 3.2 can be summarized with the following equation:

$$loss_{kl,batch} = \frac{1}{d_l} \sum_{l=0}^{d_l-1} E_{x\_values}[\log q(z_l|w_{batch}, x_{batch}) - \log p(z)], \tag{3.19}$$

where d_l is the latent space dimension and x_{values} is a discrete set of x-coordinates. It defines where the secondary kl divergence is being evaluated. In its essence, the primary approach penalizes $\boldsymbol{q(z|w_i, x_i)}$ whereas the secondary approach penalizes $\boldsymbol{q(z|w_{batch}, x_{batch})}$, which supposedly is a better approximation of $\boldsymbol{q(z)_{true}}$. This difference is visualized in figure 3.14. The red bars signify the difference at the location of evaluation, which is given by the sampled $\mathbf{z_l}$ for the primary $loss_k$ and by x_{values} for the secondary $loss_k$.

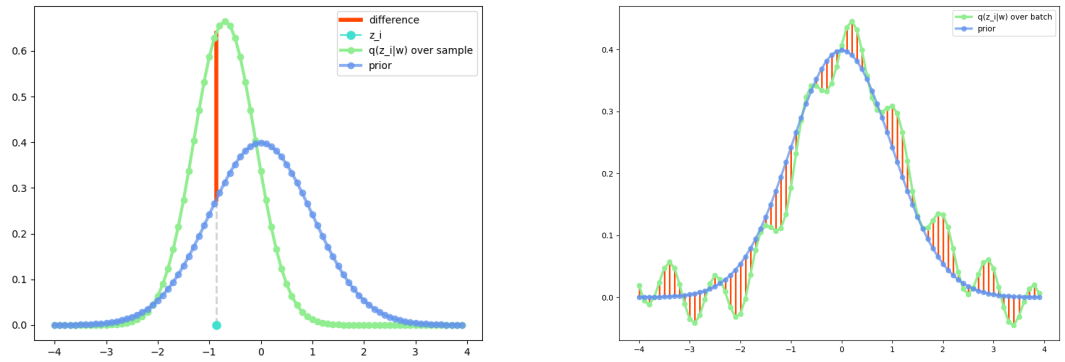


Figure 3.14: Primary (left) and secondary approach (right) to the KL loss.

The number of x_{values} is given by the parameter *kl steps* and the boundary values of this set of discrete values by *kl x*, which by default is set equal to four. If kl steps is set to zero the primary $loss_k$ is enabled. For all other positive non-zero integers, the secondary $loss_k$ is employed.

3.4.4 Metrics

$loss_{tot}$ defined in equation (3.10) is the metric of how the model is evaluating its own performance and subsequently of how it adjusts its weights and biases. Therefore it can be considered the most important metric in this thesis. Nonetheless, there is a need for additional metrics that capture a different set of characteristics and features of the model in comparison to $loss_{tot}$.

Reconstruction Error

The *reconstruction error* metrics have one important feature that differentiates it from $loss_{tot}$, they are indifferent to the *lambda* weights. This allows a comparison of model performance with different sets of lambda weights. Without the reconstruction error metrics, this would not be possible, since models with the same performance can have vastly different $loss_{tot}$ values, depending on lambda. The reconstruction error metrics are defined as

$$\begin{aligned} rec\_err_w &= \frac{1}{n} \sum_{i=0}^{n-1} |\hat{w}_i - w_i|, \\ rec\_err_x &= \frac{1}{m} \sum_{j=0}^{m-1} |\hat{x}_j - x_j|, \\ rec\_err_{tot} &= rec\_err_w + rec\_err_x. \end{aligned} \tag{3.20}$$

Since all performance attributes are normalized on a $[0, 1]$ range, the x error can directly be interpreted as a relative error percentage, e.g. an error of 0.1 is equivalent to a 10% error. The w error needs to be divided by two before this intuitive transformation is applicable, as the pixel range is $[-1, 1]$.

Design Error

The *Design Error* is responsible for evaluating the models ability of generating sound panels conditioned on $\mathbf{x}$. The design error itself is simply the difference of the estimated $\hat{\mathbf{x}}_{\mathbf{gen}}$ of the generated panel $\mathbf{w}_{\mathbf{gen}}$ and $\mathbf{x}$. The procedure for the computation of $\hat{\mathbf{x}}_{\mathbf{gen}}$ is shown in listing 3.3.

Listing 3.3: Generate panel and compute estimated performance attribute.

```
def compute_x_hat_gen(x):

# 1. sample latent vector from normal distribution
z = normal(zeros(z_dim), ones(z_dim))

# 2. forward z and x through decoder to generate panel
w_gen = decoder.forward(z, x)

# 3. forward generated panel through x reconstruction pipeline of encoder
x_hat_gen = encoder.reconstruct_x(w_gen)

# 4. return estimated x of generated panel
return x_hat_gen
```

In order to give variability to the evaluation of the design error, there are different versions of it that are defined in equation (3.21).

$$\begin{aligned} design\_err &= |\hat{x}_{gen} - x|, \\ design\_err_{n_{gen}} &= \frac{1}{n_{gen}} \sum_{i=0}^{n_{gen}-1} |\hat{x}_{gen,i} - x|, \\ design\_err_{n_{best}/n_{gen}} &= \frac{1}{n_{best}} \sum_{i=0}^{n_{best}-1} |\hat{x}_{gen\_best,i} - x|, \end{aligned} \tag{3.21}$$

where $\hat{x}_{gen\_best}$ are the estimated performance attributes of the n_{best} best panels from a total of n_{gen} generated panels. These different versions of the design error are indicated by its subscripts. $design\_err$ is the design error of a single generated panel from x and computationally the least expensive to calculate. $design\_err_5$ is the average of five generated panels and $design\_err_{3/10}$ is the average of the three best from a total of ten generated panels, where the best panel has the lowest $design\_err$.

If the hypothesis posed in chapter 1 does not hold, it might be interesting to experiment adding a *design loss* to the loss function in equation (3.10), that behaves similar to the design error. The design loss is defined as

$$loss_d = \frac{1}{n} \sum_{i=0}^{n-1} (\hat{x}_{gen,i} - x_i)^2, \tag{3.22}$$

where n is the batch-size and the generation of $\hat{\boldsymbol{x}}_{\boldsymbol{gen,i}}$ is given in listing 3.3. It is not trivial to train the model on the design loss in the first place, since the model could find loop holes in order to minimize the design loss without actually solving the design problem. The model would only focus on encoding the requested $\boldsymbol{x}$ in the output image $\hat{\boldsymbol{w}}$, without an incentive of generating an image resembling a sound panel. Therefore, a *sleep* and *transition* parameter is introduced

in combination with λ_d. It allows to gently introduce the λ_d weight with *sleep* and *transition* periods defined as

$$\lambda_{d,current} = \min(\frac{\max(epoch_{current} - sleep + 1, 0)}{transition}, 1)\lambda_d, \tag{3.23}$$

where $\lambda_{d,current}$ is always in the range $[0, \lambda_d]$. With these additions, the $loss_{tot}$ defined in equation (3.10) is adjusted to

$$loss_{tot} = \lambda_w loss_w + \lambda_x loss_x + \lambda_{kl} loss_{kl} + \lambda_{d,current} loss_d. \tag{3.24}$$

Mixed Error

Since the design error itself is not a good measure for overall performance and the reconstruction only indicates the panel generation performance conditioned on x, there is a need for an all-encompassing, global metric. Therefore, the *mixed error* is introduced, which combines the reconstruction and the design error, as the name suggests. It is defined as

$$mixed\_err = rec\_err_{tot} + 0.25 \cdot design\_err. \tag{3.25}$$

The *design_err* is scaled down because the design performance is *only* successful if combined with a small rec_err_{tot}. Therefore, it is to be avoided that a small *design_err* alone can yield a respectable *mixed_err*. Otherwise it is considered a fluke and the small *design_err* is produced by exposing a loophole in the design task. A scale of 0.25 is a good compromise of the *design_err* being relevant and of being confident that the *mixed_err* can not be minimized without a good performance in the classic reconstruction tasks.

3.4.5 Optimizer

To adjust the weights after backpropagating the loss, the **Adam** optimizer provided by pytorch is used. The optimizer is initialized by configuring the *learning rate* and the *weight decay* parameters. In order to decouple the weight decay λ_{wd} from the batch-size and number of epochs, the formula given in equation (3.26) is applied, which was proposed in [8].

$$\lambda_{wd,norm} = \lambda_{wd}\sqrt{\frac{b}{n_{samples} \cdot n_{epochs}}}, \tag{3.26}$$

where b is the batch-size. The weight decay functions in a similar manner as a classic L_2 regularization. It reduces the models tendency of overfitting and enables it to generalize better by a gradual decay of the models weights. However, it is not desirable to apply this regularization to batch norm layers, therefore these weights are excluded. Another methodology of increasing the models performance on validation and test data, is to introduce weight *dropouts* in the networks, which were introduced in table 3.5. Technically they are not configured by the optimizer, but they belong to the same domain of optimizing model performance. Instead of decreasing the magnitude of the weight, a probability p_{drop} is introduced with which the weight is being nullified. This probability is given by the dropout parameter and will only be applied on feed-forward networks. Since the reconstruction of $\boldsymbol{w}$ and $\boldsymbol{x}$ are two separate tasks, the parameters *weight decay x* and *drop x* act only on the **xffn** layer, whereas *weight decay w* and *drop w* apply to the rest of the network. Table 3.8 summarizes the parameters that were introduced in this section.

reconstruction	kl divergence	design error	optimizer
lambda w	lambda k	lambda d	lr
lambda x	kl batch	sleep	weight decay w
lambda g	kl steps	transition	weight decay x
log likelihood	kl x		drop w
rel loss			drop x

Table 3.8: CVAE loss parameter definitions.

3.5 Supplementary Data Representations

As in most cases, the images in our dataset are only an abstraction of the actual objects they represent. Encoding the z-coordinate as the pixel value enables us to display a 3d object in a 2d environment. With this knowledge, a 3d model of a sound panel and its corresponding 2d image can be considered as analogous from an information point of view. However, the neural network does not have this knowledge. All it sees is the image and the performance attributes and the rest is up to the network to explore. Allowing the model to be exposed to additional meaningful data representations of the same sound panel, can help advance its understanding of the underlying patterns and relationships.

3.5.1 FFT Filter

The *Fast Fourier Transform* (fft) is a popular method in deep learning and computer vision [3]. It transforms the input from the real into the frequency domain. We make use of it by filtering the sound panel image for its low- and high-frequency contents, decomposing $\mathbf{w}$ into $\mathbf{w_{dec}}$. By doing so, the task of reconstructing $\hat{\mathbf{w}}$ is divided in multiple but less complex sub-tasks $\hat{\mathbf{w}}_{\mathbf{dec}}$, since each image only contains a certain range of frequencies. The assumption is that the $\mathbf{w_{dec}}$ will help the model to reconstruct especially the high-frequency contents of the images, as well as filtering out relevant characteristics that define the performance attributes. The operational procedure of decomposing the image $\mathbf{w}$ into its sub-images is detailed in listing 3.4.

Listing 3.4: Decomposition of image.

```
def decompose_w(w, filters):

    # 1. apply the fast fourier transform on w
    w_fft = fft(w)

    # 2. iterate over filter ranges and decompose w in the frequency domain
    w_fft_dec = []
    for filter in filters:
        w_fft_filtered = filter.apply(w_fft)
        w_fft_dec.append(w_fft_filtered)

    # 3. transform w decomposed back into real domain
    w_real_dec = []
    for w_fft in w_fft_dec:
        w_real = fft_inverse(w_fft)
        w_real_dec.append(w_real)

    # 4. return decomposed w
    return w_real_dec
```

The number and magnitude of filters are defined by introducing two new parameters named $filter_1$, $filter_2$ and $filter_3$. They define the boundaries within the frequency domain to create the *ranges* with which the input image $\mathbf{w}$ is decomposed. The normalized ranges are defined as

$$\begin{aligned} filter\_range_1 &= [0, \quad filter_1], \\ filter\_range_2 &= [filter_1, filter_2], \\ filter\_range_3 &= [filter_2, filter_3], \\ filter\_range_4 &= [filter_3, \quad 1]. \end{aligned} \tag{3.27}$$

If all filters are set to zero, there is only a single filter range for $[0, 1]$, meaning no filtering at all. An example is shown in figure 3.15, with two non-zero filters and therefore three active filter ranges. The corresponding frequency domains are given in figure 3.16. The color respectively grey-scale range of each image is normalized separately for a clearer visualization.

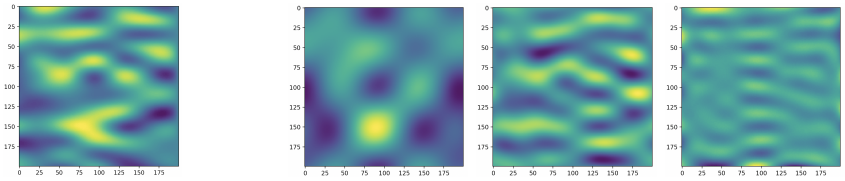


Figure 3.15: Image $\mathbf{w}$ left, decomposed images $w_{dec,0}$, $w_{dec,1}$ and $w_{dec,2}$ on the right.

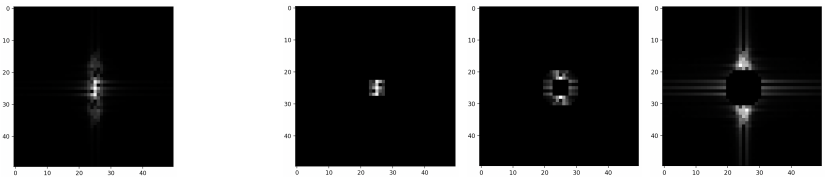


Figure 3.16: Corresponding frequency domains (cropped).

In order to forward the set of decomposed images $\mathbf{w_{dec}}$ through the model, the number of channels for each cnn is multiplied times the number of sub-images. The concept is analogous to RGB images, only that in this case frequency domains are being passed through the cnn network instead of color channels. Similarly to RGB channels, the reconstructed decomposed output images $\hat{\mathbf{w}}_{\mathbf{dec}}$ are combined to create the final output image $\hat{\mathbf{w}}$. In order to further optimize the fft filter approach, two new parameters *avg xffn* and *dec loss* are introduced that allow slight modifications in the method execution.

parameter	value	
avg xffn	True	$\hat{x} = xffn(wz_{avg})$
	False	$\hat{x} = xffn(wz_{dec})$
dec loss	True	$loss_w = sum(mse(\hat{w}_{dec}, w_{dec}))$
	False	$loss_w = mse(\hat{w}, w)$

Table 3.9: Definition of avg xffn and dec loss parameters.

As shown in table 3.9, if avg xffn is set to *True*, $\boldsymbol{wz_{dec}}$ is averaged over its decomposition before being passed to the xffn network. This reverts the increase in dimensionality introduced by the number of decomposed images n_{dec} for $\boldsymbol{wz_{avg}}$. When avg xffn is set to *False*, $\boldsymbol{wz_{dec}}$ is regularly forwarded to xffn. If dec loss is set to *True*, $loss_w$ is computed on $\hat{\boldsymbol{w}}_{\boldsymbol{dec}}$ and therefore on each decomposed image separately before being summed up. In the case of dec loss being *False*, $\hat{\boldsymbol{w}}_{\boldsymbol{dec}}$ is summed up over its sub-images to create $\hat{\boldsymbol{w}}$ before computing $loss_w$.

3.5.2 FFT CNN

Building on the framework that was laid out in section 3.5.1, the images in the frequency domain shown in figure 3.16 might have more value than just being used as a decomposition tool of $\mathbf{w}$. For this reason, a new network **fft cnn** is introduced, which takes a cropped version of $\boldsymbol{w_{fft}}$ defined in listing 3.4 as input. The resulting output is forwarded solely to the xffn network, under the assumption that it is not a meaningful data representation for the reconstruction of $\boldsymbol{w}$. The output image of cnn fft is defined as $\boldsymbol{wx_{fft}}$, or

- $\boldsymbol{wx_{fft}} \leftarrow p_{cnn_{fft}}(wx_{fft}|\boldsymbol{w_{fft}})$.

The cnn is constructed using the already defined blocks. Two additional parameters are necessary to finalize the fft cnn, which are defined in section 3.5.2.

parameter	
n blocks cnn fft	gives the number of cnn blocks to be used.
dim fft	defines the dimension of the cropped $\boldsymbol{w_{fft}}$ image.

3.5.3 Slicer

It is questionable how well the model is able to take a 2d input and extract features that are relevant in a 3d topography. The objective of the *slicer* method is to supply an additional data representation of the sound panel that enables the model to develop an enhanced three-dimensional intuition. This is achieved by slicing the sound panel at different heights and creating a low resolution 2d binary image that abstracts the shape of the panel at that specific z-coordinate, as visualized in figure 3.17.

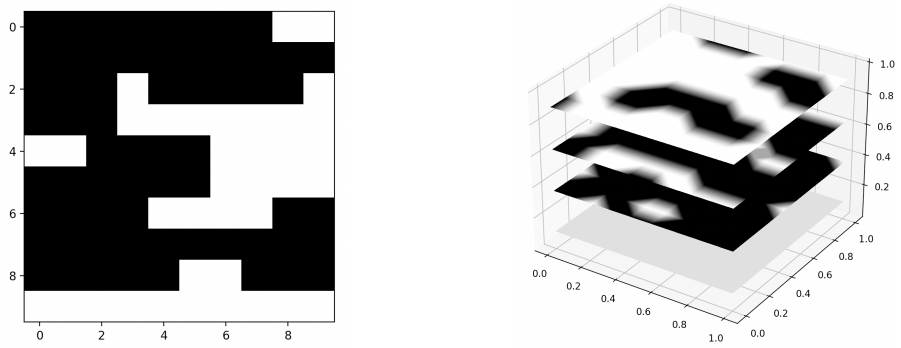


Figure 3.17: Single slice (left) and stacked slices (right).

It is assumed that these slices enhance the perception of a three dimensional space and create an intuitiveness about the *depth* of the sound panels. Once the number of desired slices is set, the z-coordinates are defined by evenly spacing the interval $[min(w), max(w)]$ of the given panel, where the resulting image of the first and the last slice is trivial and therefore discarded. The desired resolution of the slice is attained by applying a simple maxpool operation on $\boldsymbol{w}$. The slices are flattened to create a one dimensional vector $\boldsymbol{s}$ and concatenated to $\boldsymbol{wz}$ before being passed into the ffn networks. The number of slices is given by *n slices* and the desired resolution by *slice dim*. In order to further modify the slicer methodology, two additional parameters are introduced in section 3.5.3.

parameter		
slice xffn	True	$\hat{x} = xffn(wz + s)$
	False	$\hat{x} = xffn(wz)$
slice zffn	True	$z = zffn(wz + s)$
	False	$z = zffn(wz)$

These two additional sub-processes of the fft cnn and slicer pipeline are implemented in the existing model as shown in figure 3.18.

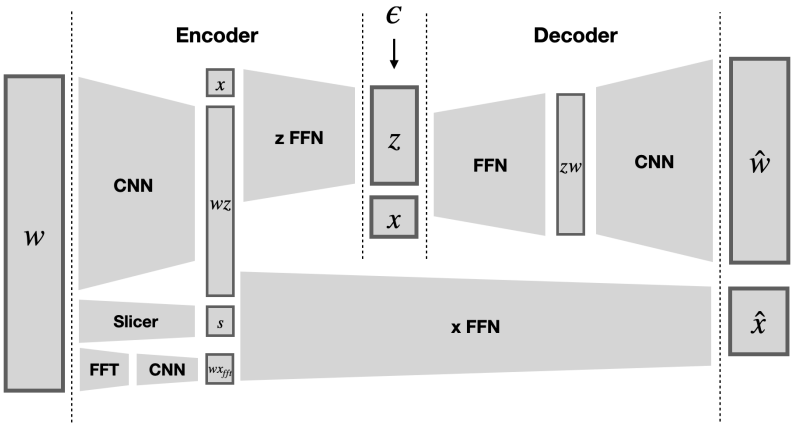


Figure 3.18: CVAE model architecture with additional sub-networks.

Table 3.10 summarizes the parameters that were introduced in this section.

fft filter	fft cnn	slicer
filter 1	n blocks cnn fft	n slices
filter 2	dim fft	slice dim
filter 3		slice xffn
avg xffn		slice zffn
dec loss		

Table 3.10: supplementary data representation parameter definitions.

Chapter 4

Results

In the results section, all relevant runs of the model are documented, visualized and its performance evaluated. It provides a structured overview of the models evolution and clarifies how decisions have been made regarding the parameters listed in table 4.1. The base, tuned and final model are presented with selected plots and figures, whereas the graphics of the hyperparameter tuning sessions are held to a minimum. The appendix contains a complete set of figures for the presented models, as well as a visualization of each tuning session. The plots and figures have been created using *Weights & Biases*, *matplotlib* and *plotly*.

4.1 Base Model

Model		**FFT Filter**		**FFT CNN**		**Slicer**	
enc zffn norm	True	filters	(0, 0, 0)	n blocks cnn fft	0	n slices	0
enc xffn norm	True	avg xffn	False	dim fft	0	dim slice	0
dec cnn norm	True	dec loss	False			slice xffn	False
cnn block	21					slice zffn	False
net design	regular						
z dim	128						
Losses		**Regularization**		**Optimizer**		**Design Error**	
lambda w	1	drop w	0	optim	Adam	lambda d	0
lambda x	1	drop x	0	lr	5e-4	sleep	0
lambda g	0	weight decay w	0	batch-size	256	transition	0
log likelihood	False	weight decay x	0	epochs	30	lambda k	1e-3
rel loss	False					kl steps	0

Table 4.1: Parameter configuration of the base model.

In order to evaluate the concepts and methodologies demonstrated in chapter 3, a *base model* is introduced that functions as a performance bar. It is the most basic setup of the CVAE model and gradually more functionalities are being added in section 4.2. The base parameters are set as given in table table 4.1.

Network	sub net	input dim	dim1	dim2	dim3	dim4
Encoder	cnn	(1, 200, 200)	(8, 98, 98)	(16, 47, 47)	(32, 22, 22)	(64, 9, 9)
6.1M par	zffn	5184	512	(2, 128)		
	xffn	5184	512	6		
Decoder	ffn	134	512	5184		
3.1M par	cnn	(64, 9, 9)	(32, 22, 22)	(16, 47, 47)	(8, 98, 98)	(1, 200, 200)

Table 4.2: Components of the base CVAE model.

The dimensions of the models sub-networks are detailed in table 4.2. The model contains a total of 9.2M parameters, taking up an estimated memory of 36MB. The encoder and decoder are very similar, with the large exception being the added xffn in the encoder.

4.1.1 Performance

The table below details the performance of the base model in various metrics. The total and design error are highlighted as they are the main basis for comparison between different models.

evaluation	epoch	train loss	val loss	total error	design error
best	27/30	0.0084	0.0124	**0.1012**	**0.1937**

When observing the losses chart displayed in figure 4.1, the *val x loss* stands out negatively. It stagnates rather quickly whereas the *train x loss* is consistently improving. The model has obvious problems of generalizing the attribute reconstruction, which is accounting for the largest share of the validation loss as a consequence. The train loss is linearly improving on a logarithmic scale, showing great potential for the model. If the val x loss can be generalized better, the overall results will improve significantly.

Figure 4.1: Timeline of loss and error progression.

The errors on the right plot are approximately mirroring the val losses. The patterns are very similar, albeit the progression curves are generally slightly flatter, which can be traced back to the quadratic nature of the MSE. However, a concerning development is that the design error does not seem the follow the patterns of the other errors, displayed in the right table. It shows some overall improvement, but the trendline is very flat. Its high dependency on an accurate attribute reconstruction could be a substantial factor in these results. Potentially, improving the x error should have a significant impact on the design error, assuming the hypothesis stated in the introduction is to hold.

Reconstruction

For a qualitative comparison between different models, a similar panel range as presented in figure 3.8 is computed from validation data and reconstructed for the attributes *fft mean* and *high res y*. The selected images translate into a small but fairly representative sample size of the dataset.

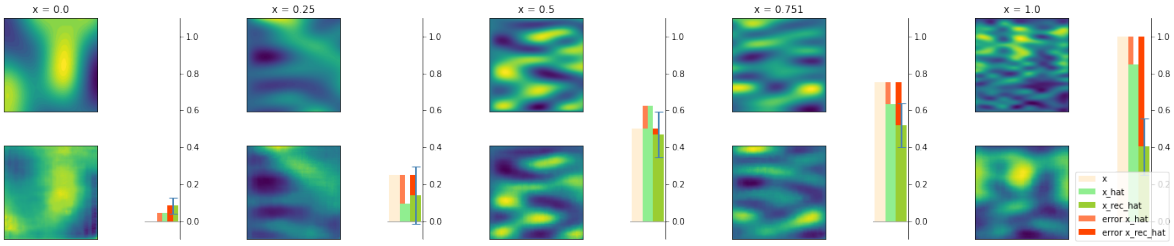


Figure 4.2: Panel reconstruction and fft mean attribute errors.

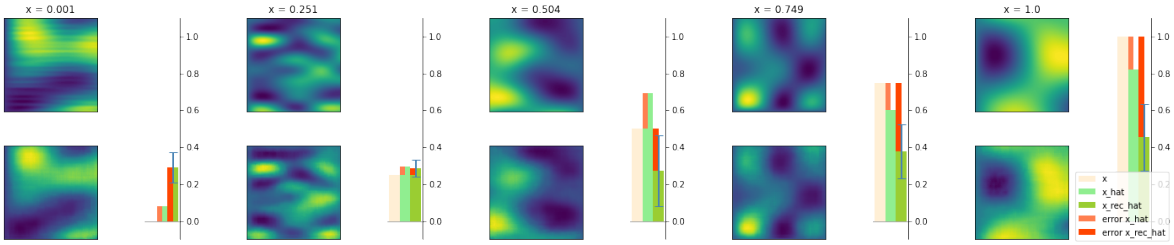


Figure 4.3: Panel reconstruction and high res y attribute errors.

The upper panels represent the inputs, the lower panels the reconstructed outputs. The model already is doing a respectable job with reconstructing the panels, considering that 40′000 parameters are being encoded in a latent space of 128 variables. As shown in the legend on the bottom right, the figure displays the true attribute value $\boldsymbol{x_i}$ in *beige* (left bar) as well as the reconstructed $\hat{\boldsymbol{x}}_{\boldsymbol{i}}$ in *light green* (middle bar) and its corresponding attribute error in *light red* (on top of middle bar). The attribute reconstruction is slightly off the mark for some examples, reflecting the performance presented in figure 4.1. However, it is able to approximate the true attribute. The estimated attribute $\hat{\boldsymbol{x}}_{\boldsymbol{rec,i}}$ for the reconstructed panel is given in *green* (right bar) and its corresponding error in *red* (on top of right bar). It is defined as $|\hat{x}_{rec,i} - x_i|$ and the first application of the full reconstruction and evaluation loop. In order to compute this error, the model reconstructs the panel and forwards $\hat{\boldsymbol{w}}$ to the attribute reconstruction pipeline of the encoder. Subsequently, it can measure its own performance by computing the *estimated* performance attribute $\hat{\boldsymbol{x}}_{\boldsymbol{rec,i}}$ of the reconstructed panel and compare it with the initial $\boldsymbol{x_i}$. The approximated uncertainty of this estimation is visualized with the blue error bars on top of the $\hat{\boldsymbol{x}}_{\boldsymbol{rec,i}}$ bar, which are equal to the value of the original attribute reconstruction error. The $\hat{\boldsymbol{x}}_{\boldsymbol{rec,i}}$ error is a good first indication of the models ability to generate panels on conditionality. These figures suggest that the model will struggle with the conditional generation of panels, as the red error bars are consistently large. A complete set of these evaluations can be found in the appendix figure A.2.

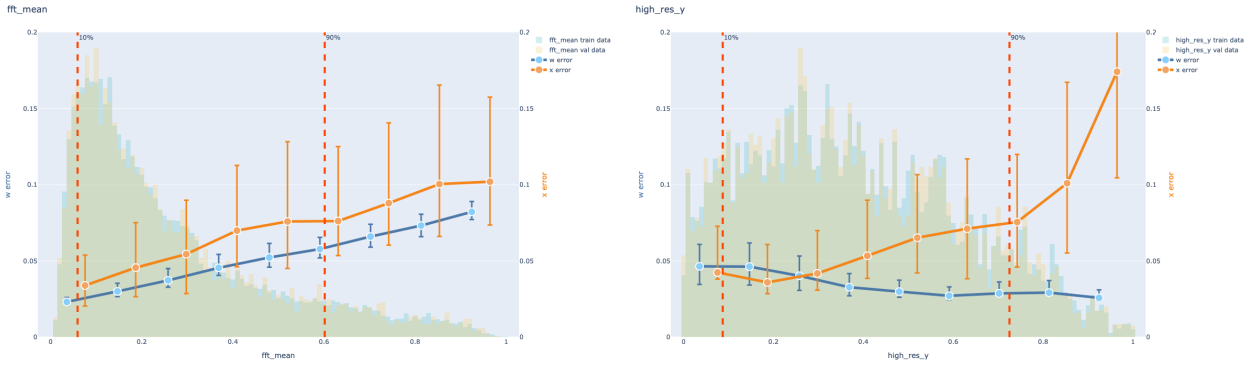


Figure 4.4: Base error distribution of fft mean and high res y attributes.

A more quantitative evaluation of the reconstruction performance is presented in figure 4.4. Each plot analyzes the full distribution of a performance attribute and splits the data in ten buckets, corresponding to a relative range of 10% at a time. From each bucket, the mean of the $\boldsymbol{w}$ and $\boldsymbol{x}$ reconstruction errors is calculated and plotted, where the error bars represent the 25% and 75% confidence intervals, respectively. The blue line represents the error of $\boldsymbol{w}$ reconstruction, the red line the $\boldsymbol{x}$ reconstruction. In figure 4.4, the evaluated attributes are *fft mean* (left) and *high res range* (right), the full set of figures is documented in figure A.1. As expected, the errors are peaking where the data is least representative, especially in the case of the $\boldsymbol{x}$ error for the high res attribute. The average error for the final bucket (or datapoint in the plot) is close to 20% and the 0.75 confidence interval even above that mark.

Latent Space

In order to reveal the hidden latent space distribution, two separate methods have been implemented, as visualized in figure 4.5. Initially, a large set of computed $\boldsymbol{\mu_z}$ and $\boldsymbol{\sigma_z}$ are collected by passing the validation data through the encoder. The distribution of this collection is visualized in the left plot, where the dashed lines represent the regularizing prior. The blue line displays how $\boldsymbol{\mu_z}$ is distributed, the green line $\boldsymbol{\sigma_z}$. On the right plot, each of the 128 $\boldsymbol{z}$ variables are estimated by numerically adding up and normalizing the resulting distributions from the collected $\boldsymbol{\mu_z}$ and $\boldsymbol{\sigma_z}$, where the red dashed line represents the regularizing prior. This procedure is part of the secondary $loss_k$ computation described in equation (3.19), only that the *batch* is the full validation set in this case.

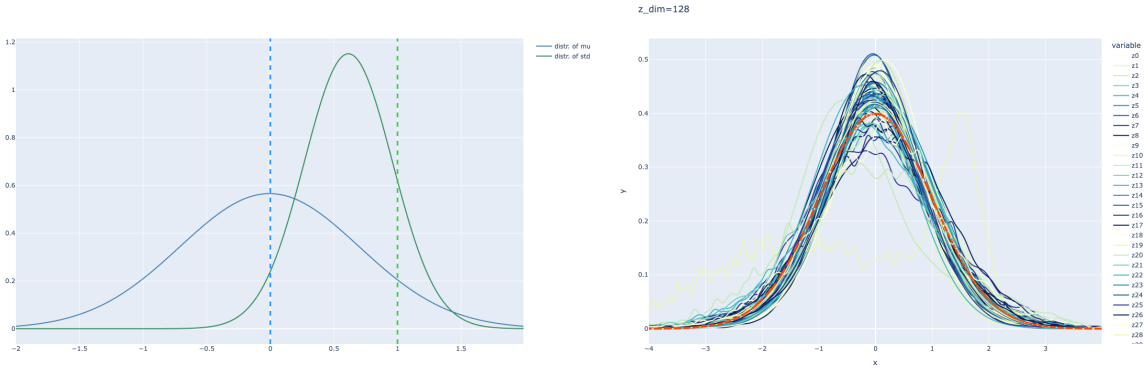


Figure 4.5: Distribution of mu and sigma (left) and estimated latent space variables (right).

The results give another intuitive representation of the kl divergence's role in a CVAE model. As evident in the left plot, the separate $\boldsymbol{\mu_z}$'s and $\boldsymbol{\sigma_z}$'s have diverged significantly from the prior and are accordingly penalized. When inspecting the right plot, the estimated distributions of the latent space variables resemble a standard normal Gaussian with some minor modifications. This means that the kl divergence has done its job in regularizing $\boldsymbol{\mu_z}$ and $\boldsymbol{\sigma_z}$. The latent space in its entirety is still embodying the desired properties.

Design Error

While the $\hat{\boldsymbol{x}}_{\boldsymbol{rec}}$ is a good first estimation of the final objective, figure 4.6 is the final assessment of the models panel generation and self evaluation performance. It is structured similarly as figure 4.4, with the only difference being that the red line represent the $design/err_{10}$ and the light blue line the $design/err_{3/10}$, as defined in equation (3.21). The x-axis indicates the design conditions that are passed to the model, where the $\boldsymbol{x}$ attributes of the full validation data set are used as attribute requests. The figures are scaled to an error of 0.7 or 70%, and shockingly the averaged design error reaches that mark. By further observation, one can conclude that the model produces designs with approximately the same attributes, no matter the request. That is evident by the fact that the design error increases almost linearly from a certain point in the distribution. The low point in the design error plot is the preferred attribute value of the models generated designs. It is safe to conclude that the panel generation conditioned on performance attributes is not working yet.

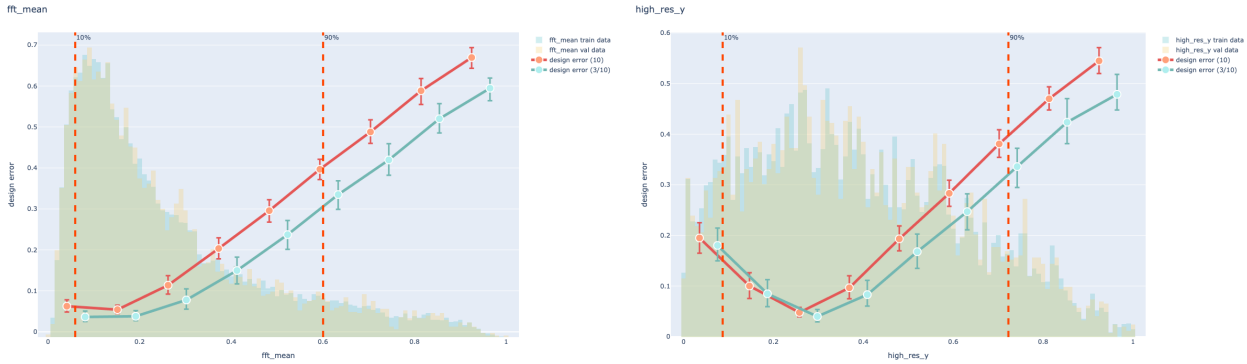


Figure 4.6: Base design error of fft mean and high res y attributes.

4.2 Hyperparameter Tuning

The hyperparameter tuning process should provide clarity and gradually reduce the number of free parameters. The model is still resemblent of a black-box, not much is known regarding its performance and potential at this point. Each run reveals a glimmer of information that builds up an understanding of the models dynamics. Only if the hyperparameter tuning is carried out carefully and with attention to detail, a more efficient and interpretable model can be realized. However, this process is not exact science. Decisions are frequently made on assumptions and intuition instead of based on absolute certainty. It is a trade-off between efficiency and confidence, where the sweet-spot is at neither boundary.
As seen in table 4.1, there is a large number of parameters that are yet to be implemented and defined. Optimizing for all of these parameters at once is computationally infeasible. Thus, the parameters are split in a set of subcategories, where the tuning *sessions* are being carried out. These parameter subsets are already given in table 4.1. After each tuning session, the default parameters from the base model are adjusted with the best performing configuration and passed to the next subcategory. The tuning sessions are executed out as so-called *Sweeps*, which is an internal tool of the *Weights and Biases* platform. It allows for the configuration of the relevant

metric and parameters, as well as the search space and search method. The used methods are *grid*, *random* and *bayes*. Two mechanisms of early-stopping have been implemented in the tuning setup, the *hyperband* algorithm and a classical early-stopping callback. The former compares its own performance with previous runs, whereas the latter is only concerned on its own progression. The minimum number of epochs before the early-stopping evaluation is enforced needs to be defined for both. For practical reasons, the two parameters are given as $n_{epochs,1}/n_{epochs,2}$.

4.2.1 Model

The first hyperparameter tuning session is documented in additional detail to guide the reader through the process. Here, the parameters given by the subcategory *Model* are split again and tuned separately in *session A* and *session B*. Initially, the run is configured by setting the basic metrics. In the case of *session A*, given by table 4.3, the batch norm parameters are being evaluated. The search space is given by $[True, False]$ for all three parameters and the applied method is *grid*, which yields a total of eight combinations. By setting the number of runs to eight, the whole search space is explored.

session config A		parameter	search space	set	evaluation	
method	grid	enc zffn norm	True, False	True	train loss	0.0099
epochs	10	enc xffn norm	True, False	False	**val loss**	**0.0121**
early-stopping	5/5	dec cnn norm	True, False	False	total error	
runs	8					

Table 4.3: Summary of model design tuning A.

The execution of the tuning session is visualized in figure 4.7, which generally is documented in the appendix A.2. It displays the parameters of each run and the corresponding performance. The color coding is useful in order to identify the most influential nodes in the defined search space. In order to build intuition of the models dynamics, it is just as valuable to understand which configurations do not result in a good performance as those that do. After evaluating the results, each parameter is *set* as displayed in the parameter section of table 4.3. The losses and errors of the best performing run are shown in the *evaluation* section, where the metric that is used by the early-stopping and search space algorithms is given in bold.

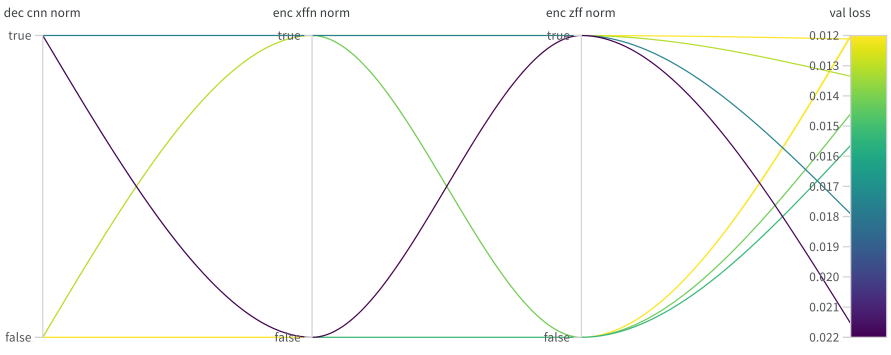


Figure 4.7: Overview of batch norm tuning.

For session A, the most influential parameter is clearly the decoders cnn norm batch layer. That is already evident in figure 4.7, but even more clearly visible if singled out and visualized, as shown in figure 4.8. The *lines* represent the mean of the given group of runs, whereas the *body* defines the groups boundaries.

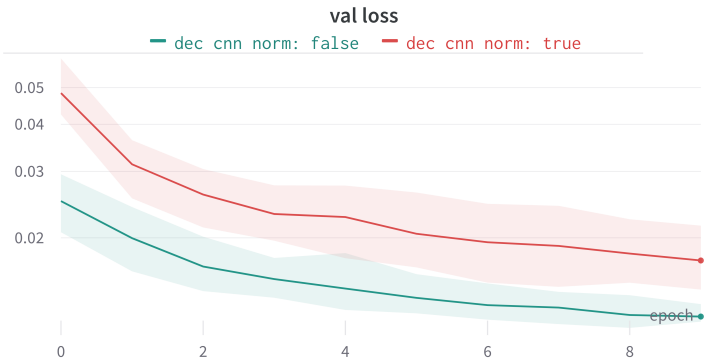


Figure 4.8: Significance of dec cnn norm parameter.

The evaluations have shown that the batch norm layers for the decoder cnn and the encoder xffn should be disabled. This is an intuitive result, since those are the two output layers for the reconstruction of panels and attributes. Especially in the case of the decoder cnn, it is harder to optimize the reconstructed panel if the cnn blocks rescale their outputs. Interestingly, it is preferred to keep the batch norm layer of the cnn zffn network active. $\boldsymbol{z}$ is not a final output of the network, except for the computation of $loss_k$, which explains the preference of keeping the batch norm layer active. However, the results were not close of being conclusive in this case.

session config B		parameter	search space	set	evaluation	
method	grid	cnn block	21, 22, 32	21	train loss	0.0092
epochs	10	design	regular	regular	**val loss**	**0.0116**
early stop	5/5		regular ffn		total error	
runs	27		deep			
		z dim	64, 100, 128	128		

Table 4.4: Summary of model design tuning B.

Session B tunes the high-level parameters of the model structure. All parameters have a discrete set of three values, yielding a total search space of twenty-seven combinations. The model is still relatively basic and efficient, thus it is reasonable to explore all twenty-seven configurations, as defined in the config section of table 4.4. The overview of tuning session B is shown in figure 4.9.

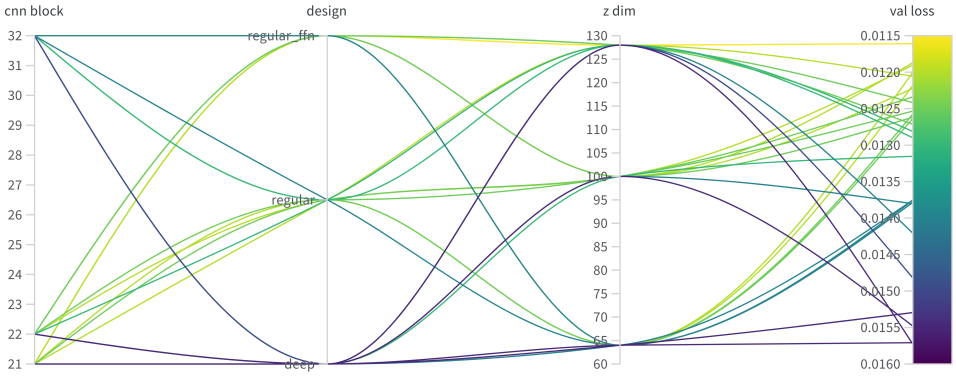


Figure 4.9: Overview of network design tuning.

What stands out most from the results is that in its current form, the model generally performs better with a simple architecture. This is the case for the complexity of the cnn blocks as well as for the network depth. The parameter with the biggest impact is the net design. In figure 4.10, it is evident that the *deep* network does not perform well in comparison to its counterparts. However, these results need to be taken with a grain of salt since only the first ten epochs are evaluated. Generally, ten epochs is enough to discern a trend that holds true also for a longer training period. In the case of the net design, the deep network can be excluded from further conversation with fairly high confidence. The remaining two network designs *regular* and *regular ffn*, that only differentiate in the layer size of the ffn blocks, perform almost identically. Thus, the simpler design parameter is chosen as there seems to be no reason to increase the number of neurons of the ffn nets. As for the latent space dimension *z dim*, it is not surprising that a larger latent space results in better performance. An increased bottleneck allows for more information to flow through. A latent space dimension of 128 is still ambitious considering the complexity of the panels, therefore it is reasonable to go with the highest performance and not change the latent space size. This concludes the model tuning and finalizes the network architecture.

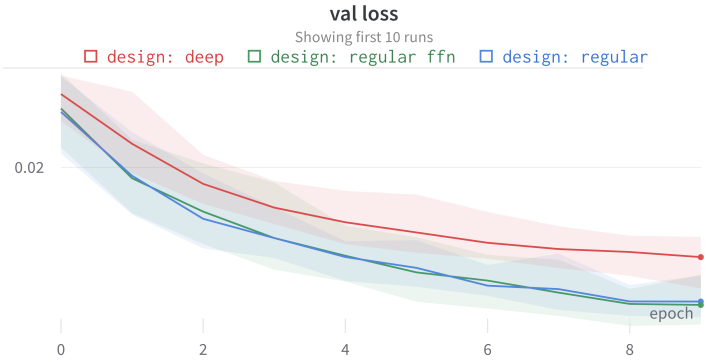


Figure 4.10: Significance of design parameter.

4.2.2 Slicer

session config A		parameter	search space	set	evaluation	
method	grid	slice xffn	True, False	True	train loss	0.0085
epochs	10	slice zffn	True False	False	**val loss**	**0.0112**
early stop	-	n slices		5	total error	
runs	4	slice dim		5		

Table 4.5: Summary of slicer tuning A.

In *session A* of the slicer tuning, default values have been set for the *n slices* and *slice dim* parameters. This enables an evaluation of the slicer in general and whether the resulting output is useful for both, or either one of the encoders ffn networks, before optimizing the number of slices and their respective dimension. Figure (c.1) in appendix A.2.1 shows an improvement of the validation loss if the slices are forwarded to the xffn network. On the contrary, the slices clearly decrease the performance if forwarded to the zffn network. This is to be expected, since the *depth* of the panel is not necessarily helpful in order to reconstruct information that is already fully available at the input. As a matter of fact, there is arguably no information source that can help the model to reconstruct the panel, other than the panel itself. Table 4.5 summarizes the setup and results of the first slicer tuning session.

session config B		parameter	search space	set	evaluation	
method	random	slice xffn		True	train loss	0.0076
epochs	10	slice zffn		False	**val loss**	**0.0105**
early stop	5/5	n slices	$[3, 10] \sim uni$	4	total error	
runs	20	slice dim	$[5, 12] \sim uni$	11		

Table 4.6: Summary of slicer tuning B.

The learnings of session A are now applied in *session B*, by setting the slice xffn and zffn values to *True* and *False*, respectively. The search space of the n slices and slice dim parameters is distributed uniformly and discretely over the ranges given in table 4.6, as only integer values are allowed. The given search space yields forty-nine different combinations and twenty of those have been sampled randomly by setting the metric to *random* and conducting twenty runs. The execution of the tuning session is shown in figure (c.2) of appendix A.2.1. Four slices at a dimensioniality of (11, 11) turned out to be the best configuration of the sliced data representation of the panel. This yields a binary vector $\boldsymbol{s}$ of size (484) that is being concatenated to $\boldsymbol{wz}$ before being passed to the xffn network.

4.2.3 FFT Representations

FFT Filters

session config A		parameter	search space	set	evaluation	
method	grid	avg xffn	True, False	True	train loss	0.0085
epochs	10	dec loss	True False	True	val loss	0.0102
early stop	-	filter 1		0.01	**total error**	**0.0910**
runs	4	filter 2		0		
		filter 3		0.03		

Table 4.7: Summary of fft filter tuning A.

Similarly to the slicer tuning, there are two boolean parameters that are defined prior to tuning the actual values of the filters. The parameters *avg xffn* and *dec loss* modify the application of the fft filter method, as defined in table 3.9. As the dec loss parameter might have a small impact on the scaling of val loss, the total error is used as the guiding metric for this tuning session. Both of the modifications introduced by the two parameters seem to improve the performance of the model with the filters set to default values of (0.01, 0, 0.03). Especially aggregating the three frequency channels into a single channel before passing it to the xffn layer seems to have a significant impact on general performance, as demonstrated in the left plot of figure 4.11.

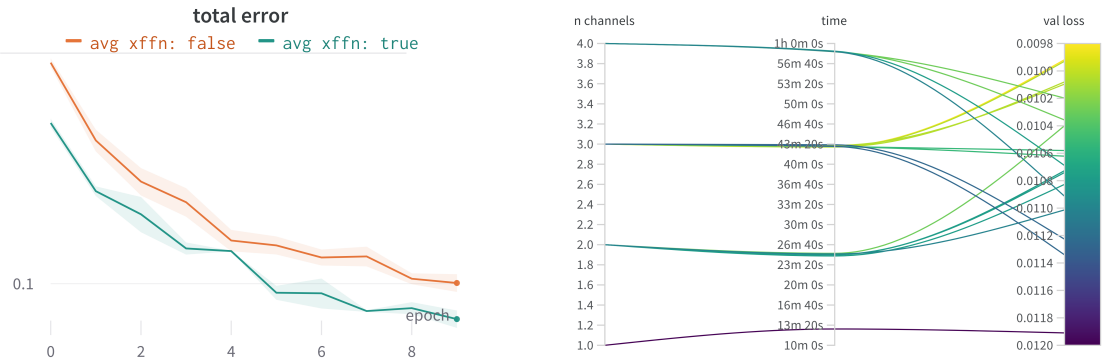


Figure 4.11: Significance of avg xffn parameter (left) and impact of channels on computation time (right).

session config B		parameter	search space	set	evaluation	
method	grid	avg xffn		True	train loss	0.0080
epochs	10	dec loss		True	**val loss**	**0.0099**
early stop	5/5	filter 1	0, 0.01, 0.015	0.015	total error	0.0900
runs	27	filter 2	0, 0.02, 0.025	0		
		filter 3	0, 0.03, 0.035	0.03		

Table 4.8: Summary of fft filter tuning B.

In *session B* of the fft filter tuning, each filter is given a discrete search space of three possible values, where one of them is zero. By using the grid method and testing all possible configurations with twenty-seven runs, not only can we evaluate the individual filter values but the impact the number of frequency channels has on performance. The right plot in figure 4.11 gives a compressed overview of the session B, where the focus is set on the number of channels. The figure suggests that three channels and therefore two active filters improve the model the most. The value of these filters has high relevancy as well, evident by the spread of the separate runs with three frequency channels. Additionally, the plot gives insights in the computation time, which turns out to be linearly increasing with the number of channels.

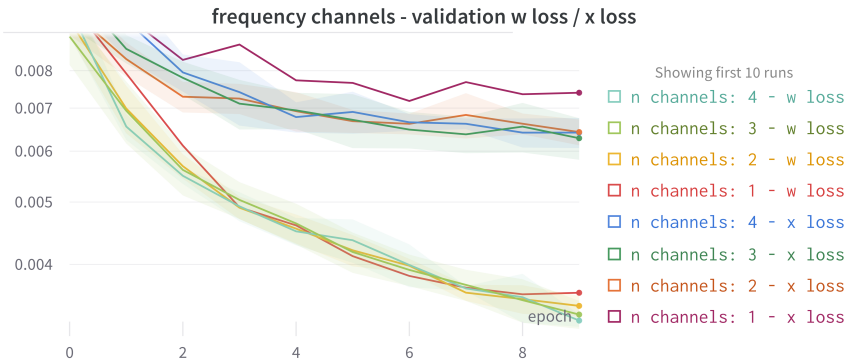


Figure 4.12: Significance of frequency channels regarding w loss (bottom) and x loss (top).

In figure 4.12, the impact of the frequency channels on $loss_x$ and $loss_k$ is investigated. Surprisingly, the panel reconstruction seems to be unaffected by the decomposition of $\boldsymbol{w}$, it does not increase

nor decrease performance. The attribute reconstruction on the other hand benefits from the frequency filters, as the run with a single channel displays the worst performance. In order to stay computationally efficient, the filters are disabled for the ensuing tuning sessions and activated once the tuning has concluded.

FFT CNN

session config		parameter	search space	set	evaluation	
method	grid	fft dim	30, 40	40	train loss	0.0101
epochs	10	n fft cnn blocks	1, 2, 3	2	**val loss**	**0.0108**
early stop	5/5				total error	0.0937
runs	6					

Table 4.9: Summary of fft filter tuning.

The search space for the two parameters yields a total of six possible configurations, which is explored in its entirety. The corresponding overview of the session can be found in figure (a.1) in the appendix A.2.2. It is worth noting that even though the validation loss in table 4.9 increased compared to the performance in the fft filter tunings, the comparison is flawed as the filters have been disabled as previously mentioned. An additional test with enabled filters showed an overall improvement with a **val loss** of **0.0096**, as documented in figure (a.2) in appendix A.2.2. Therefore, the model seems to be able to extract valuable information from the frequency representation of the sound panel. The fft cnn tuning session concludes the modifications that are made on the model itself and the focus shifts towards optimization in the following sessions.

4.2.4 Losses

session config		parameter	search space	set	evaluation	
method	bayes	lambda x	[0.1, 1] ∼ uni	1	train loss	0.0103
epochs	10	lambda w	[0.1, 1] ∼ uni	1	val loss	0.011
early stop	5/5	lambda g	[1e-3, 1e-1] ∼ log uni	0.01	**total error**	**0.092**
runs	10	log likelihood	False, True	False		
		relative loss	False, True	True		

Table 4.10: Summary of reconstruction losses tuning.

In the tuning session of the reconstruction losses, the *bayes* algorithm is used for the first time. The search space of *lambda x* and *w* is given by a uniform distribution whereas *lambda g* is uniformly distributed on a logarithmic scale. This enables the algorithm to virtually decrease *lambda g* to zero or boost its significance up to 0.1. Additionally, the parameters that enable or disable the *log likelihood* and the *relative loss* are included in the same tuning session. In this setup, the validation loss is useless as the bayes algorithm would simply decrease all weights to its minima in order to minimize the loss. Thus, the lambda-agnostic total error is employed to evaluate the runs and guide the tuning session. The regularizer *lambda k* is not included in this session, as its regularization purpose is not captured by the reconstruction loss nor error. The smoothness of the latent space becomes relevant once the design error is evaluated, therefore its introduction is postponed. There are no apparent surprises regarding the results of the lambda weights, except that a small value for the gradient loss had a favorable influence on the total error.

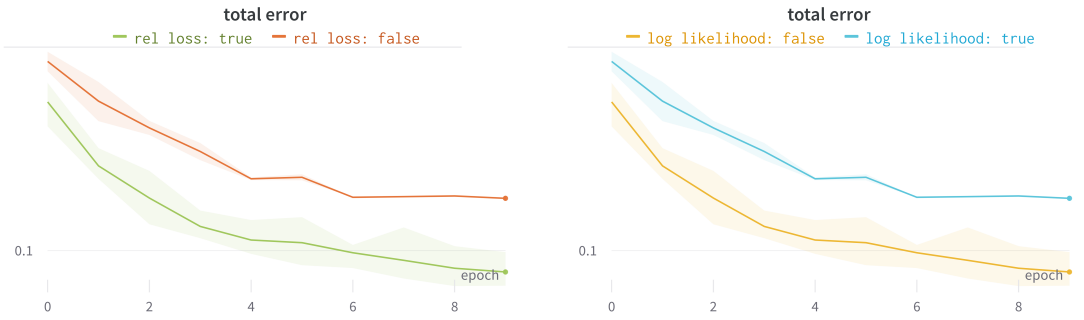


Figure 4.13: Significance of rel loss (left) and log likelihood (right) parameters.

The result for the relative and the log likelihood loss are both rather convincing, as demonstrated in figure 4.13. By closer inspection, one can observe that the parameters have not been mixed by the bayes algorithm in any of the runs, meaning *rel loss: False* is always matched with *log likelihood: True* and vice versa. Figure (b) in appendix A.2.2 confirms this observation. This leads to two identical plots in figure 4.13. This is one of the shortcomings of the bayes algorithm, it tends to ignore certain combinations. However, since the results can be compared to the previous session where both parameters were set to *False*, the suggestions in figure 4.13 can be confirmed. Apparently, the mse loss is outperforming the log likelihood loss significantly. It is important to note that the relative loss has rescaled the w reconstruction loss and therefore the validation loss can not directly be compared with previous models.

4.2.5 Dropout and Weight Decay

session config A		parameter	search space	set	evaluation	
method	bayes	drop w	[1e-4, 3e-1] ∼ log uni	1e-3	train loss	0.0050
epochs	30	drop x	[1e-4, 3e-1] ∼ log uni	2e-3	val loss	0.0098
early stop	15/5	weight decay w	[1e-4, 1e-2] ∼ uni	1e-3	**total error**	**0.081**
runs	25	weight decay x	[1e-4, 1e-1] ∼ uni	1e-2		

Table 4.11: Summary of dropout and weight decay tuning A.

When examining the losses of the base model shown in figure 4.1, the potential to improve the val x loss by implementing *dropout* and *weight decay* is supposedly high. The session A is configured with a dropout probability range that goes up to 0.3 and a maximum number of epochs of thirty. However, as shown in the overview figure (c) of the tuning runs in appendix A.2.1, the bayes algorithm is very hesitant to increase the dropout probabilities. This is also reflected by the conservative dropout values that are *set* by the best performing configuration in table 4.11.

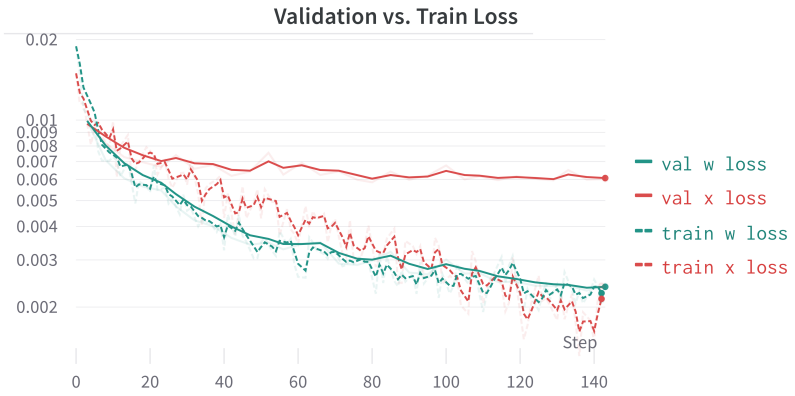


Figure 4.14: Comparison of validation and train losses.

Further investigation on the best performing run shows a large discrepancy of the train and validation x loss, as displayed in figure 4.14. Given the low values for the dropouts chosen by the bayes searching method, these results are not surprising. In order to understand the dropout dynamics better, an additional test run is initialized.

session config B		parameter	search space	set	evaluation	
method	grid	drop w	0.01, 0.05	1e-3	train loss	0.0058
epochs	50	drop x	0.01, 0.02, 0.05, 0.1	2e-3	val loss	0.0094
early stop	30/10				**total error**	**0.079**
runs	8					

Table 4.12: Summary of dropout and weight decay tuning B.

The number of epochs in session B is increased to fifty and the early stopping to 30/10, in order to let the runs develop without interruption. The search method is switched from bayes to grid to make sure that the higher dropout probabilities are properly tested. Table 4.12 summarizes this setup.

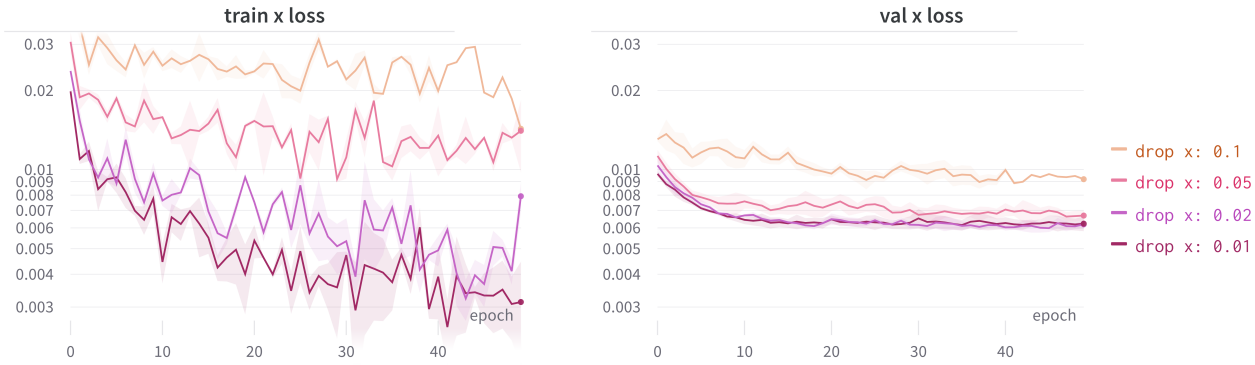


Figure 4.15: Overview of dropout and weight decay tuning.

The plots in figure 4.15 group the runs of the four drop x probabilities together and visualizes the train and val x loss over fifty epochs. These results confirm that the attribute reconstruction is very sensitive to dropouts. Even a rather insignificant probability of 5% has a massive impact,

especially on the train loss. It is not clear what the root cause of this sensitivity is. It seems that the attribute reconstruction is a complex task that suffers from small perturbations in the train loss and the disruption does not lead to a better generalization in the validation loss. Even though session B produced slightly better final results, it was also trained for almost double the number of epochs. Projecting the results of session A for an additional twenty epochs is likely to improve the results even further, therefore the *set* parameters are not adjusted in table 4.12.

4.2.6 Optimizer

session config		parameter	search space	set	evaluation	
method	bayes	optim	Adam, AdamW	Adam	train loss	0.0078
epochs	15	batch size	64, 128, 256, 512, 1024	128	**val loss**	**0.0106**
early stop	7/3	lr	[1e-5, 1e-3] ~ log uni	4e-4	total error	0.0857
runs	20					

Table 4.13: Summary of optimizer tuning B.

In the optimizer tuning, the number of epochs is reduced back to fifteen, since the evaluation is no longer focused on generalization and overfitting. The tuning session is using the bayes algorithm to explore the search space of the discretely distributed *batch size* and log uniform distributed *learning rate*, while a comparison of *Adam* and *AdamW* is conducted. AdamW is an advanced version of Adam, with an improved application of weight decay. It is always advisable to investigate how the own model reacts to these adjustments, before taking a decision on the optimizer type. The twenty runs are visualized in figure (d) in appendix A.2.2.

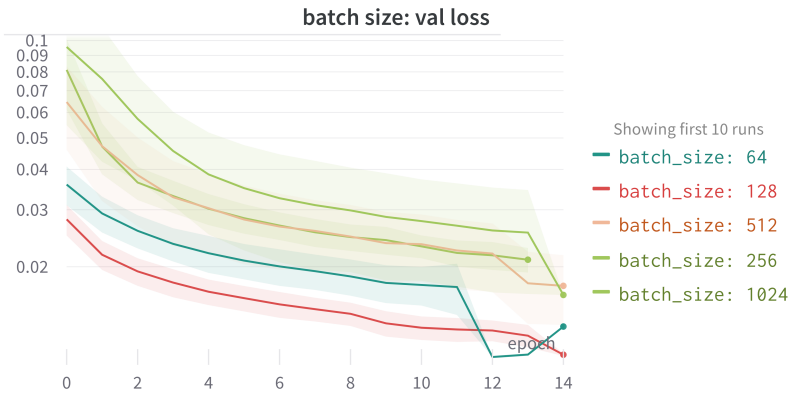


Figure 4.16: Significance of batch size.

The most notable development in this tuning session is the performance of the different batch sizes. Figure 4.16 groups the validation loss of the tested batch sizes, where the body is given by the standard error. Surprisingly, the best batch size is not the largest, but a relatively small batch of 128 samples. Most literature suggests that the larger the batch size, the better. However, Keskar et. al. [6] demonstrate the dynamics of the batch size in deep learning and how it can be beneficial to generate sharp gradients with smaller batch sizes. The results in figure 4.16 seem to support these claims.

4.3 Tuned Model

Model		**FFT Filter**		**FFT CNN**		**Slicer**	
enc zffn norm	True	filters	**(0.015, 0.03)**	n blocks cnn fft	**2**	n slices	**4**
enc xffn norm	**False**	avg xffn	**True**	dim fft	**40**	dim slice	**11**
dec cnn norm	**False**	dec loss	**True**			slice xffn	**True**
cnn block	21					slice zffn	False
net design	regular						
z dim	128						
Rec Loss		**Regulariz.**		**Optimizer**		**Design Error**	
lambda w	1	drop w	**1e-3**	optim	Adam	lambda d	0
lambda x	1	drop x	**2e-3**	lr	**4e-4**	sleep	0
lambda g	**0.01**	weight d. w	**1e-3**	batch-size	**128**	transition	0
log likelihood	False	weight d. x	**1e-2**	epochs	30	lambda k	1e-3
rel loss	**True**					kl steps	0

Table 4.14: Parameter configuration of the tuned model.

The tuned model demonstrates the accumulation of accomplishments made in section 4.2. The parameters were optimized with the objective of minimizing the panel and attribute reconstruction error, anticipating an improved design error as a result. Table 4.14 displays the parameters that define the tuned model, with every adjustment in **bold**.

Network	sub net	input dim	dim1	dim2	dim3	dim4
Encoder	cnn	(3, 200, 200)	(24, 98, 98)	(48, 47, 47)	(96, 22, 22)	(192, 9, 9)
12.8M par	cnn fft	(1, 40, 40)	(8, 18, 18)	(16, 7, 7)		
	slicer	(1, 200, 200)	(4, 11, 11)			
	zffn	15558	512	(2, 128)		
	xffn	6368	512	6		
Decoder	ffn	134	512	15552		
9.2M par	cnn	(192, 9, 9)	(96, 22, 22)	(48, 47, 47)	(24, 98, 98)	(3, 200, 200)

Table 4.15: Components of the tuned cvae model.

The encoder of the tuned model has evolved compared to the base encoder, as shwon in table 4.15. On one hand, the number of channels of the cnn have tripled, due to the introduction of the image decomposition based on the panels frequency values. On the other hand, two new sub-networks are introduced, the slicer and the cnn fft. Except for the increase in channels, the decoder has not changed. In total, the tuned model contains 22M parameters, taking up an estimated memory of 87MB.

4.3.1 Performance

evaluation	epoch	train loss	val loss	total error	design error
best	30/30	0.0042	0.0086	**0.0748**	**0.2033**

The tuned model is improved in all of its reconstruction metrics. It is important to note that the val loss is rescaled due to changing lambdas and adjusted calculations such as the relative loss, therefore it is not directly comparable to the base models validation loss performance. The total error decreased significantly, and the difference of the validation losses would be even greater

without the rescaling. Despite all the achievements, the design error is not improved - in fact, it is slightly increased compared to the base model. The dynamics of the design error displayed in figure 4.17 are unchanged. Further evaluations of the reconstruction and design errors can be found in the appendix A.1.2. As the tuned model has not developed the ability to generate panels conditioned on the performance attributes, additional efforts are needed to achieve the objective of this thesis.

Figure 4.17: Timeline of loss and error progression.

4.4 Design Error Tuning

In order to improve the design error, an additional tuning session is initialized that is focused on this one objective.

4.4.1 Session A: Introduction of Design Loss

session config A		parameter	search space	set	evaluation	
method	bayes	lambda d	[1e-5, 1e-1] ~ log uni	2e-5	train loss	0.0082
epochs	15	sleep	0, 5, 10	0	val loss	0.0149
early stop	10/3	transition	10		total error	0.1085
runs	30				design error	0.1882
runs	30				**mixed error**	**0.1570**

Table 4.16: Summary of design error tuning A.

In the first tuning session, *lambda d* is introduced as a parameter. With this introduction, the model is not solely focused on reconstructing $\boldsymbol{w}$ and $\boldsymbol{x}$, but it is explicitly trained on the ability to generate conditioned panels. The search space of lambda d is log uniform distributed, in order to allow the bayes algorithm to operate rather unconstrained. It is guided by the *mixed error* to account for the reconstruction and the design error, which was introduced in equation (3.25). There are a lot of uncertainties with this new method, therefore it is advisable to fully explore the space of possibilities before narrowing down on a potential solutions. Figure A.12 (a) gives the overview of the separate runs. As evident in table 4.16, this first design error tuning session does not yield the desired results. The model exploits loopholes in the system and is able to improves the train loss on the cost of the validation loss, but it struggles to im both at the same time. Figure 4.18 gives a rough overview of this dilemma.

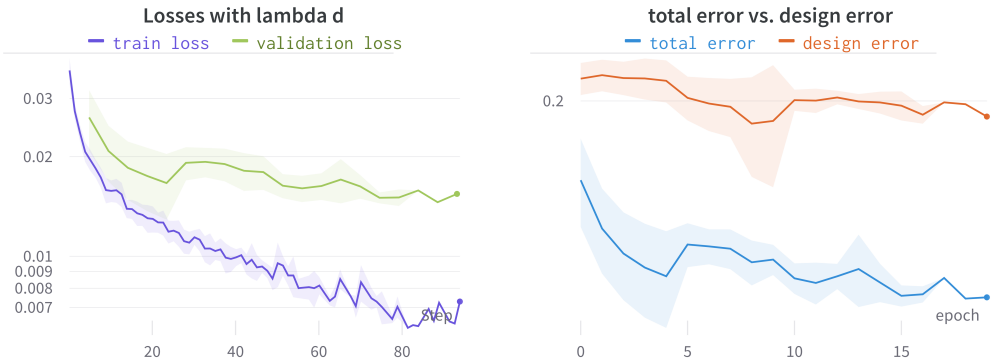


Figure 4.18: Comparison of total error and design error for tuning session A.

4.4.2 Session B: Freeze Model

The encountered problems from session A are tackled in session B by changing the approach and using the pre-trained tuned model as a starting point. In order not to allow the model to adjust the attribute reconstruction and create loopholes in the system, every sub-network that is involved in computing $\hat{\boldsymbol{x}}$ is frozen by disabling the gradients of all contained parameters. Now, the loss function contains the reconstruction of $\boldsymbol{w}$, the gradient loss, the kl divergence and the design loss, while the reconstruction of the attributes will remain constant. By introducing a new parameter *freeze*, two different modifications of the freeze method are tested. *x rec* freezes the sub-networks that are involved with $\hat{\boldsymbol{x}}$, while *extended* additionally freezes the decoder cnn, as shown in figure 4.19. In case the model is still able to bypass the actual objective, a frozen decoder cnn will stabilize $\hat{\boldsymbol{w}}$ and make it virtually impossible to trick the system.

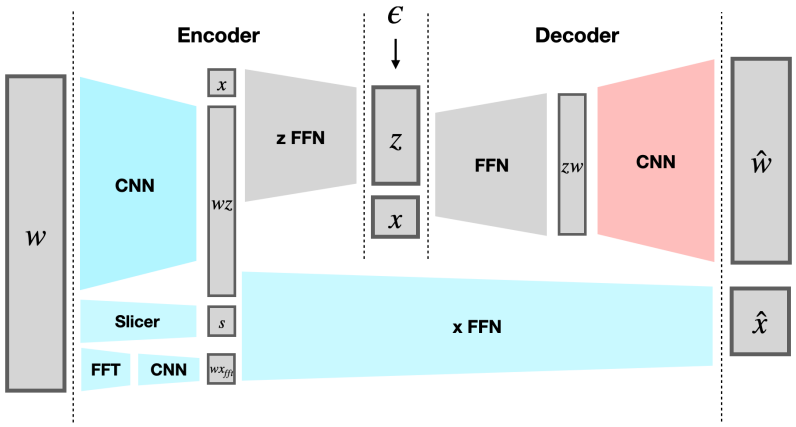


Figure 4.19: Frozen model, with the option to freeze the decoder cnn.

session config B		parameter	search space	set	evaluation	
method	grid	lambda d	1, 1e-2, 1e-4	2e-5	train loss	0.0069
epochs	15	freeze	x rec, extended	x rec	val loss	0.0097
early stop	8/3	sleep	0		total error	0.0817
runs	6	transition	10		design error	0.0824
					mixed error	**0.1022**

Table 4.17: Summary of design error tuning B.

As detailed in table 4.17, three different values for lambda d are tested with this new method. Figure A.12 (b) in the appendix details the separate runs of session B. The results suggest that the approach works as intended, the model is improving the design error significantly without losing performance in the reconstruction of panels or attributes. Apparently the model has always been capable of learning the ability to generate panels given a requested condition, but there were more efficient ways of improving the loss function. The issue was fundamentally a failure of expressing our intentions in the loss function. Figure 4.20 demonstrates the effect of freezing the decoder cnn. We can conclude that the efficiency of lowering the design error is improved if the optimizer is able to backpropagate through the sub-network.

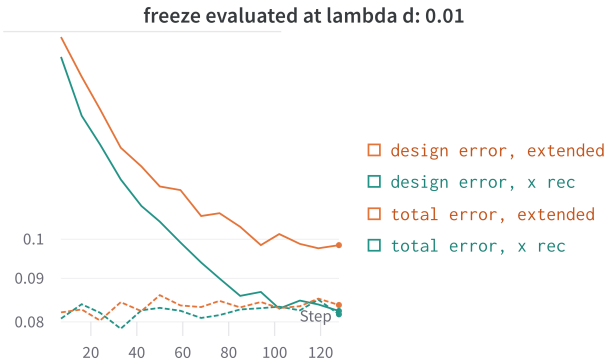


Figure 4.20: Significance of freeze parameter, where the dashed lines represent the total error.

4.4.3 Session C: Secondary KL Divergence

session config C		parameter	search space	set	evaluation	
method	grid	lambda d	0.1, 0.01	0.01	train loss	0.0069
epochs	15	kl steps	0, 40	0	val loss	0.0106
early stop	15/5	lambda k	0.01, 0.001	0.001	total error	0.0875
runs	4	transition	5		design error	0.1307
		sleep	0		**mixed error**	**0.1202**

Table 4.18: Summary of design error tuning C.

Finally it is time to test the *secondary loss k*, introduced in equation (3.19). The purpose of the kl divergence is to regularize the latent space in order to eliminate blindspots and have a smooth transition of generated panels. The benefit of the regularized latent space is not reflected in the reconstruction loss, but it is an important part of the panel generation process. If the parameter *kl steps* given in table 4.18 is set to zero, the primary loss k is employed. Otherwise, the parameter enables the secondary loss and defines the number of steps that are included in the interval of x *values*. The summary of the tuning runs are detailed in appendix figure A.12 (c). The results show that the model has a slight preference for the primary loss k and a value for *lambda k* of 1e-3, whereas the more conservative lambda d performs better. The significance of the value for kl steps and lambda d is visualized in figure 4.20.

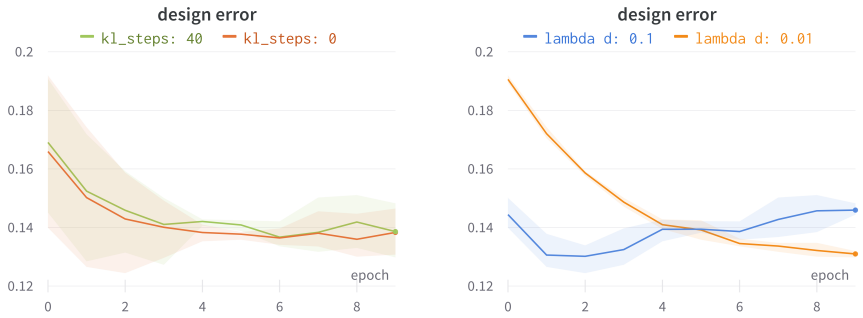


Figure 4.21: Significance of kl steps (left) and lambda d (right) parameters.

4.4.4 Session D: Dropout for Design Error

session config C		parameter	search space	set	evaluation	
method	grid	drop w	0.01, 0.1, 0.2, 0.5	0.2	train loss	0.0071
epochs	30	lambda d	0.02		val loss	0.083
early stop	5/3	transition	20		total error	0.0659
runs	4	sleep	0		design error	0.0799
		sleep	0		**mixed error**	**0.0859**

Table 4.19: Summary of design error tuning D.

The new approach of freezing a large part of the encoder invites a re-thinking of the dropout implementation. Since the sensitive $\hat{\boldsymbol{x}}$ reconstruction is no longer influenced by any dropouts, it is reasonable to experiment with less conservative and more conventional dropout values. In this final tuning session, only the *drop w* parameter is evaluated, since *drop x* does not have any influence anymore. As before, the tuned model is loaded and all x rec layers frozen. Four runs are initialized that each go over thirty epochs, to properly evaluate how the runs evolve over time. The overview of this session is given in appendix figure A.12

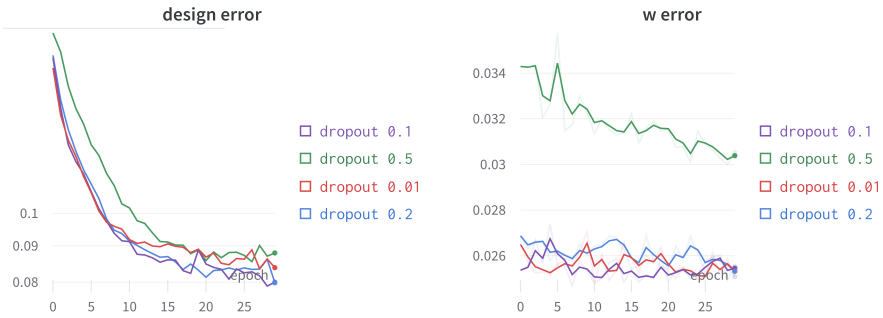


Figure 4.22: Significance dropout parameter, evaluated on design error (left) and w error (right).

The runs perform quite similarly over the thirty epochs, except for *dropout 0.5* as evident in figure 4.22. Especially in terms of the w error there is a large discrepancy, even though the error is constantly improving. However, looking at the trajectory it is a stretch to expect this run to join the performance of the others. *dropout 0.2* seems to be the most promising run, since it is constantly improving, albeit marginally. With the conclusion of this evaluation, the stage is set for the final model.

4.5 Final Model

Model		**FFT Filt.**		**FFT CNN**		**Slicer**	
enc zffn norm	True	filters	(0.015, 0.03)	n bl. cnn fft	2	n slices	4
enc xffn norm	False	avg xffn	True	dim fft	40	dim slice	11
dec cnn norm	False	dec loss	True			slice xffn	True
cnn block	21					slice zffn	False
net design	regular						
z dim	128						
Rec Loss		**Regulariz.**		**Optimizer**		**Design E.**	
lambda w	1	drop w	**5e-3|0.25**	optim	Adam	lambda d	**0|0.05**
lambda x	**1|0**	drop x	**5e-3|0**	lr	4e-4	sleep	0
lambda g	1e-2	weight d. w	1e-3	batch size	128	transition	**0|50**
log likelih.	False	weight d. x	1e-2	epochs	**100|50**	lambda k	1e-3
rel loss	True					kl steps	0
						freeze	**-|x rec**

Table 4.20: Parameter configuration of the final model.

The computation of the final model is conducted in two separate runs, with the first spanning over 100 and the second over 50 epochs. The primary run prioritizes the panel and attribute reconstruction. As for the secondary run, the encoders cnn, cnn fft and xffn layers are frozen and lambda d is introduced. The objective is to minimize the design error without increasing the w error. For the parameters in table 4.20 that have two different parameters for the two runs, they are given as $\mathbf{x_1}|\mathbf{x_2}$. Lambda d is set equal to 0.05, which seems large at first glance. Considering that the transition parameter is set to 50, lambda d is in fact equal for the first twenty epochs as the runs conducted section 4.4.4. From there, it keeps increasing linearly until the conclusion of the run. Since the model is saved at the best checkpoint at any given epoch, this setup enables to experiment with higher values of lambda d, without risking a bad final performance. The drop w parameter is set to 0.25, slightly increasing it compared to the best performing run of table 4.19, in anticipation of training for a longer period. The components of the CVAE model in table 4.21 have not changed compared to the tuned model.

Network	sub net	input dim	dim1	dim2	dim3	dim4
Encoder	cnn	(3, 200, 200)	(24, 98, 98)	(48, 47, 47)	(96, 22, 22)	(192, 9, 9)
12.8M par	cnn fft	(1, 40, 40)	(8, 18, 18)	(16, 7, 7)		
	slicer	(1, 200, 200)	(4, 11, 11)			
	zffn	15558	512	(2, 128)		
	xffn	6368	512	6		
Decoder	ffn	134	512	15552		
9.2M par	cnn	(192, 9, 9)	(96, 22, 22)	(48, 47, 47)	(24, 98, 98)	(3, 200, 200)

Table 4.21: Components of the final CVAE model.

4.5.1 Performance

evaluation	epoch	w error	x error	total error	design error	mixed error
run 1 | best	100/100	0.0261	0.0467	**0.0727**	**0.1742**	0.1157
run 2 | best	18/50	0.0286	0.0467	**0.0753**	**0.0928**	0.0985

Table 4.22: Metrics of final model.

There is a noticeable drop in performance for the errors in the final model. After some investigation, it was discovered that during the design error tuning session, the loaded model was trained on a different set of training and validation data. This falsely increased the performance on the new validation set, since the pre-trained model had already seen some of it. The inaccuracy was fixed for the final model, causing the drop of performance. The design error itself should be largely unaffected, as it was not optimized in the loaded model. Nevertheless, there is a large drop in performance of the design error which likely can be traced back to the decreased performance in attribute reconstruction. This gives some intuition on the impact of an improved x error on the design error, which seems to be considerate. Overall, the concept of training the final model in two separate runs seems to have worked well. With the first run focusing on reconstruction for 100 epochs, overfitting was successfully avoided despite low dropout rates. The second run is initialized with the new set of parameters and most of the encoder layers frozen. After the conclusion of the second run, the design error drops significantly. As a result, a final model with efficient reconstruction losses and an average estimated design error of roughly 9% is realized.

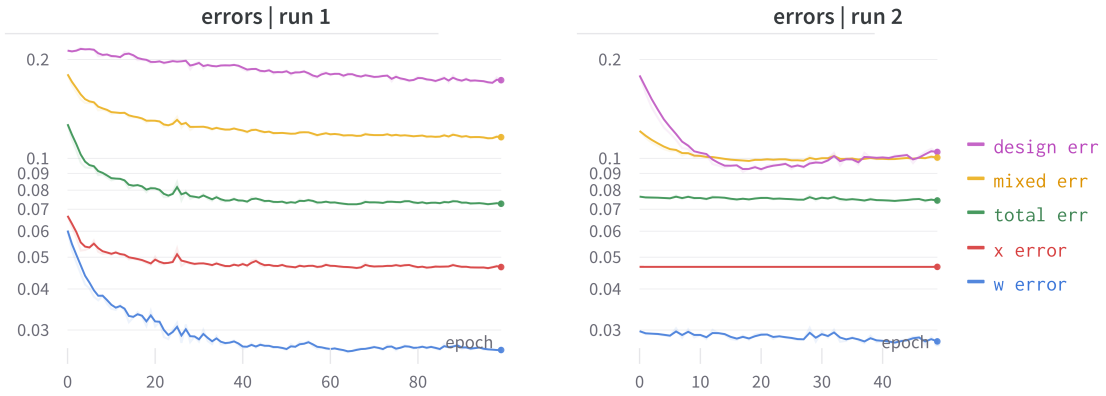


Figure 4.23: Timeline of error progression of run 1 and 2.

Due to identical scales, figure 4.23 shows a nice transition between the first and the second run. The design error in the second run starts to overfit at around epoch 15, which is another implication from the inaccuracy described above. It seems like it would benefit from a larger dropout probability, which stands in contradiction to the properties that w error shows in figure 4.22. It is not trivial to simply increase the dropout for the design error and keep it constant for the w error computation, since both relay on the same networks. Intuitively, increasing the dropouts for the decoder ffn could improve the overfitting of the design error, but it seems like a classic trade-off situation, where more research is needed before a conclusion can be reached.

Reconstruction

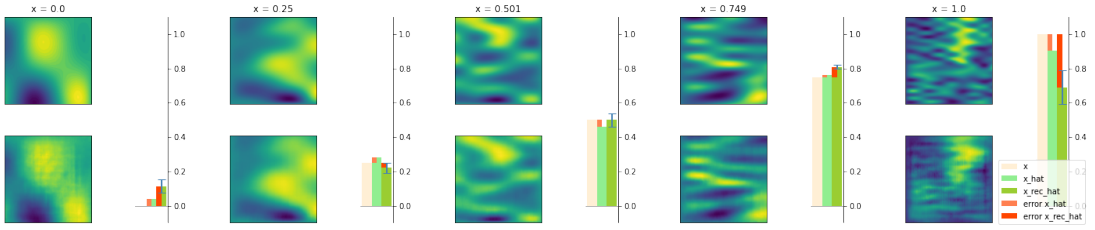


Figure 4.24: Panel reconstruction and fft mean attribute errors.

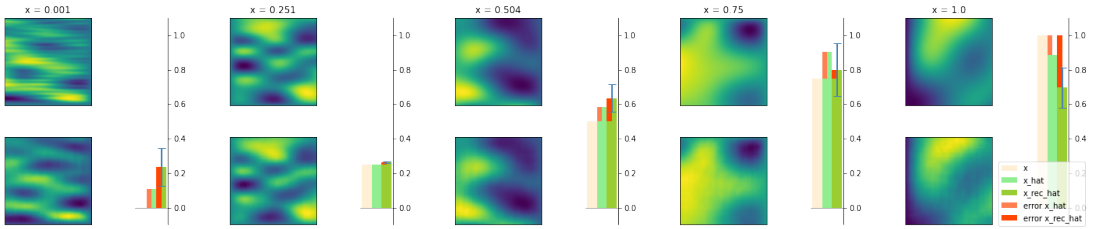


Figure 4.25: Panel reconstruction and high res y attribute errors.

The reconstruction of panels in figure 4.24 and figure 4.25 has subjectively improved, the images look smoother overall. The model still has problems with generating the high and low frequency contents of the image, as seen in the boundary examples of figure 4.24 and figure 4.25. The attribute reconstruction error of $\hat{\boldsymbol{x}}$ has clearly gotten more accurate. However, the most significant improvement is realized with the estimated attribute error of $\hat{\boldsymbol{x}}_{\boldsymbol{rec}}$. With the exception of the boundary panels, the estimated attributes of the reconstructed panels is accurate throughout the two ranges. As displayed in figure A.8 of the appendix, this improvement translates to the other attributes as well.

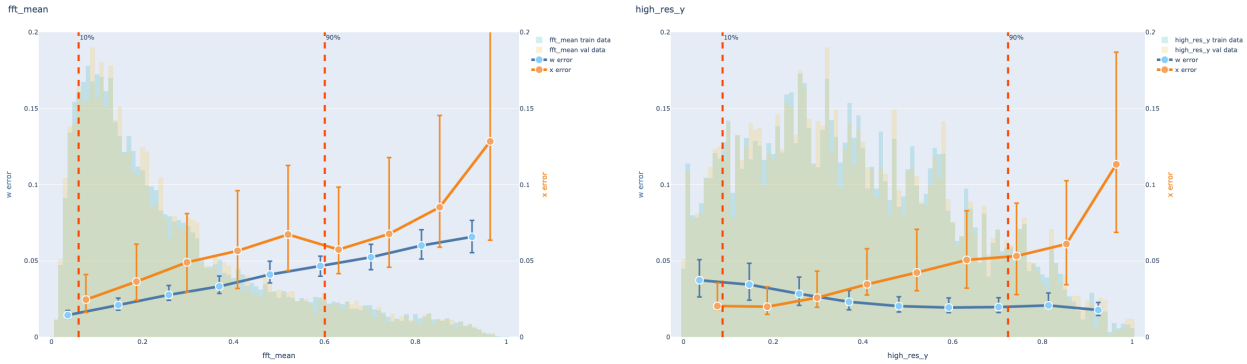


Figure 4.26: Final error distribution of fft mean and high res y attributes.

In figure 4.26 the differences are not as clearly to spot, even though the attribute reconstruction has improved overall, as was seen table 4.22. Since there are only a small amount of samples at the upper boundaries of the fft mean and high res y distributions, these large errors are viewed as insignificant by the model. It prioritizes the errors in the highly represented data buckets, where small changes have large impacts on the total loss. Thus, it is important to always be aware of the underlying data distribution when evaluating these figures, as a minimal improvement in a well represented area might weigh heavier than a large improvement at the boundary.

Latent Space

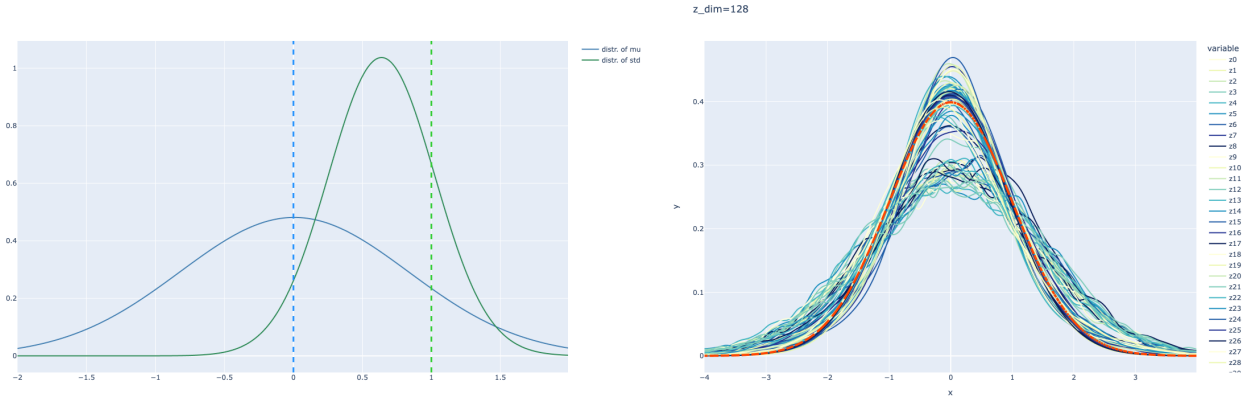


Figure 4.27: Distribution of mu and sigma (left) and estimated latent space variables (right).

The summarized latent space distribution displayed in the left plot of figure 4.27 has remained largely unchanged, which is to be expected as the kl loss has not been adjusted. It is worth noting that on the right plot, the distributions of the separate latent space variables seem to have stabilized in two sets of Gaussians. In one set, the distributions peak at a likelihood of approximately 0.4, which is the trajectory of a standard normal Gaussian. The other set peaks at an approximate likelihood of 0.3, which produces a slightly flatter distribution.

Design Error

The most important evaluation of the final model is the analysis of the design error. Figure 4.28 demonstrates a significant improvement over the base and tuned models in the royal category. The design errors have flattened given that the scales have remained constant for a proper comparison. The design error (10) is consistently below the 15% mark for the *fft mean* attribute, whereas the design error (3/10) does not go over the 10% barrier. The model performs considerably worse for the *high res y* attribute, but the design error (3/10) suggests that performance can be improved by increasing the number of panels to generate. The results seem to be consistent, given the tight 0.25 and 0.75 confidence intervals.

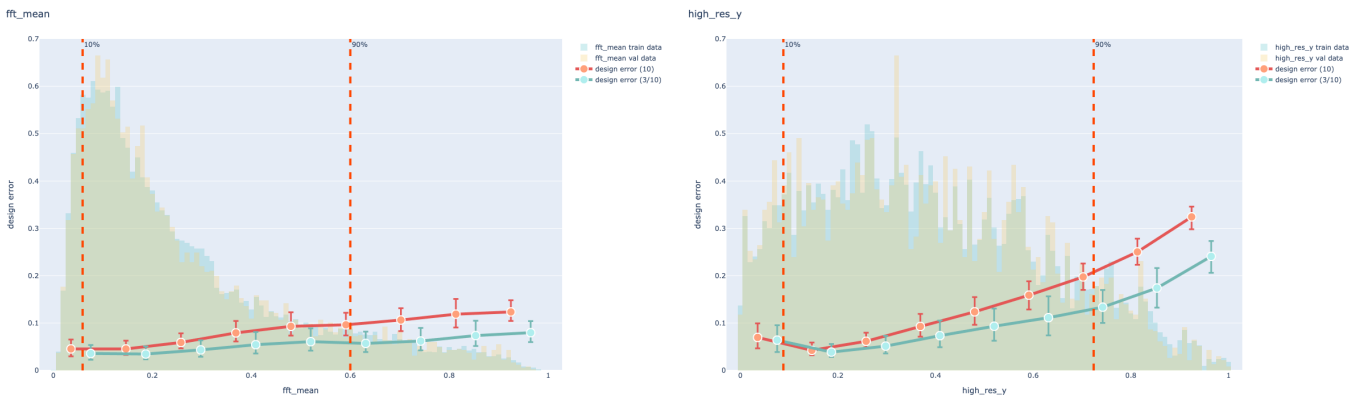


Figure 4.28: Final design error of fft mean and high res y attributes.

Testing

As a final examination of a machine learning model, it is traditionally evaluated on test data, which has not been used to train the model, to adjust hyperparameters, by early-stopping algorithms or to select the best performing checkpoints. The only connection the model has to this data is that it originates from the same dataset and therefore was generated with the same processes. Figure 4.29 shows that the model is fairly consistent with the performance of validation and test data, meaning that the tuning procedure has not lead to an overfit of the validation data and thus the generalization is not affected.

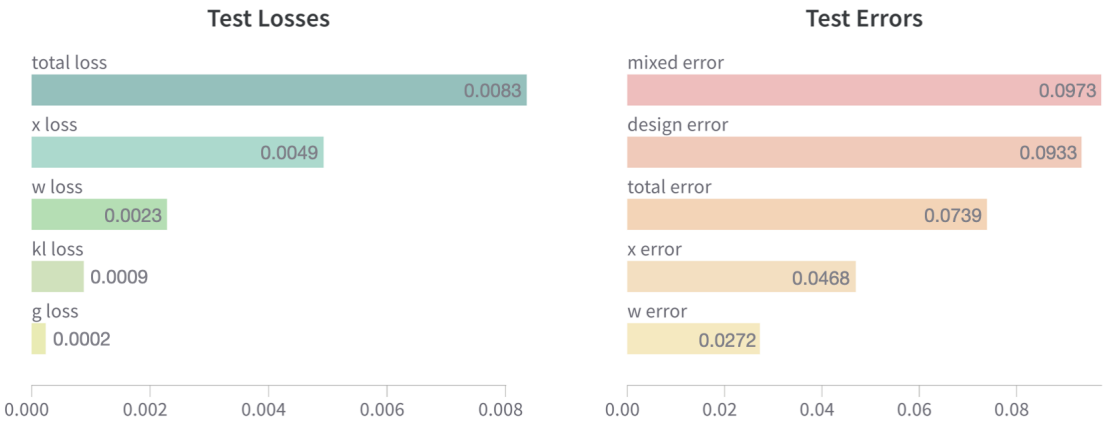


Figure 4.29: Final evaluation on test data.

4.6 Case Study: Conditional Design Exploration

4.6.1 Setting

There are fundamentally only two things needed to initiate the acoustic panel generation process: a trained decoder and a performance attribute request. The decoder is provided by simply loading the final model from its best checkpoint. The performance request is not as trivial, since the joint probability distribution of the performance attributes is unknown. Sampling from a standard distribution like a Gaussian is problematic, since a high *fft mean* value stands in contradiction to a low *fft mean x* value. Therefore, the performance attributes are randomly sampled from the test data, to make sure the request is valid. The sampled requested performance attributes are:

fft mean x	fft mean y	fft mean	panel height	high res x	high res y
0.0733	0.1278	0.0266	0.2834	0.7563	0.2479

Since generating panels is equivalent to a simple forward pass of the decoder, the computational costs are insignificant. Therefore, generating a thousand panels is a good initial baseline that can be adjusted depending on the users needs.

4.6.2 Generation and Analysis

The generated panels are modeled in a 3d environment as shown in figure 4.30, allowing the user to explore the acoustic panel landscape and gather inspirations. The models are generated using the *mayavi* library. Overall, the generated images translate into smooth structures, although slightly more rough around the edges if compared to the panels from the dataset. For some design instances there are inconsistency at the boundaries, but for the purposes of this thesis these are mostly negligible.

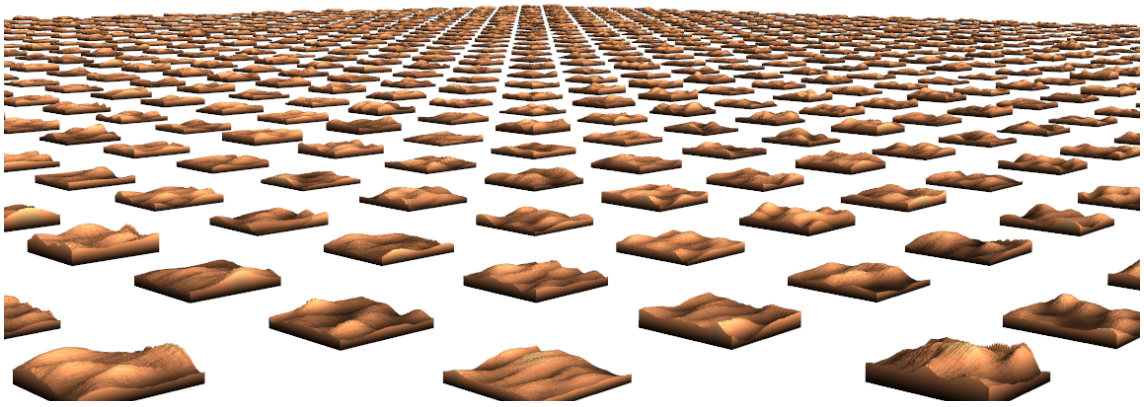


Figure 4.30: Generated panel landscape.

The average relative estimated design error of the generated panels is **7.4**%, slightly below the expected relative error of **9.3**%. Figure 4.31 gives an analysis of the generated panels and estimates their performance. The *red* histogram represents the distribution of the training data for the given attribute, the *green* histogram the distribution of the estimated attributes of the generated panels and the *blue* line is the request. The generated attributes are fairly accurate, with their respective distribution peaking at or close to the request. The lone exception is *fft mean y*, as not a single panel is estimated to be in its direct vicinity. On the contrary, the *fft mean x* attributes are estimated to be very accurate, as the distribution is peaking sharply right at the location of the request.

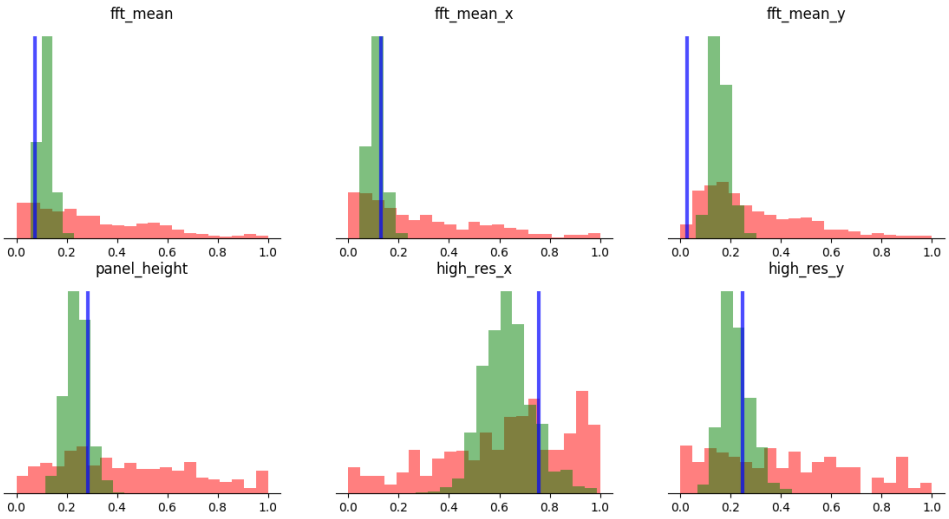


Figure 4.31: Attribute distribution of generated panels.

4.6.3 Selection

In order to get the most performance out of the generative model, a selection is made of the three best performing panels. They are displayed in figure 4.32 along with the corresponding performances attributes and the estimated design error. The attributes are are arranged as given in the table. In summary, the design error (3/1000) of the requested attributes **0.032** or in relative terms **3.2**%, which is a very accurate result. However, there is an added average uncertainty of approximately **5**%, due to the x reconstruction error. Nevertheless, performance-wise the model definitely passes the case study validation.

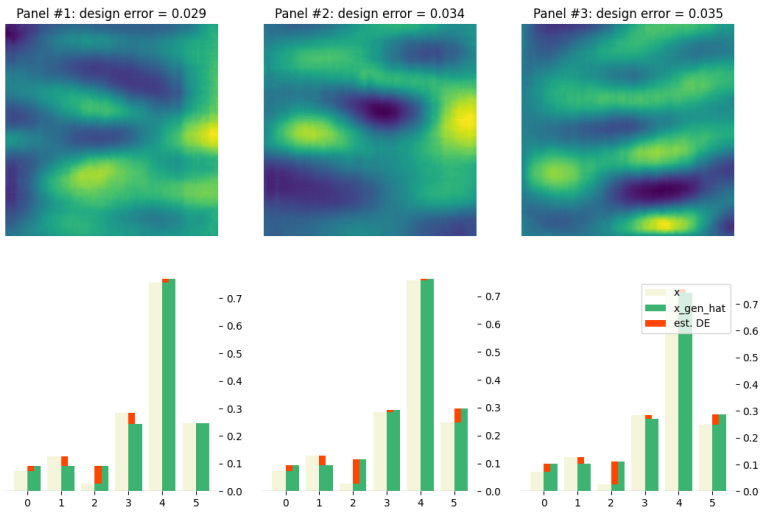


Figure 4.32: The three best panels and their respective estimated design errors.

Lastly, the three selected panels are modeled in figure 4.33. Although they do have very similar properties, the hills and valleys within the topologies are uniquely arranged. In all three cases, the texture is very smooth and there are no visible inconsistencies at the boundaries.

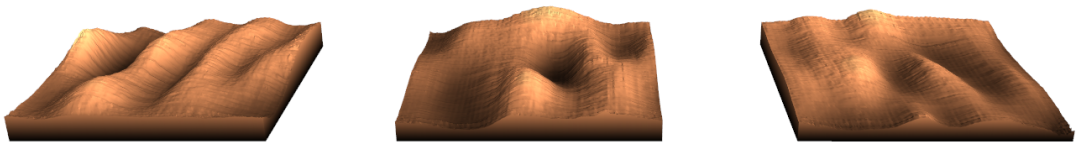


Figure 4.33: The selected panels in a 3d environment.

Chapter 5

Discussion

5.1 Achievements and Applications

Over the course of the thesis, a lot of progress has been made and a good understanding of the models dynamics has been developed. A significant amount of work went into the initial steps of pre-processing a challenging dataset and instantiating a conditional variational autoencoder that is able to learn from the data on a simple level. This laid the groundwork for the implementation of more advanced methodologies such as the fft filtering or the slicing of the panels. The demanding reconstruction of the performance attributes invited an exploration of unconventional approaches and creative thinking. These digressions have shown to consistently improve the overall performance of the model, as was thoroughly documented in the tuning process. This is not to say that all explorations have beared fruits, as for example the secondary kl divergence loss. The proposed adjustment has not resulted in an improvement, but more research on this topic would be interesting. The KL divergence, as it is implemented today, is very practical and successfully regularized the latent space in this work, but it seems that there is potential for a more elegant and optimization based formulation. There have been valuable findings along the design error exploration that ultimately resulted in a model capable of solving the design problem, as demonstrated in the case study. Although the texture of the generate panels is not perfectly smooth, it was never the intention to create a model that produces final designs. It is definitely useful to explore various possibilities given a set of requirements, if the designer is willing to accept an average estimated design error of roughly 9%. This number can be significantly lowered by generating more panels and selecting for them.

5.2 Hypothesis and Decoupling of Latent Space

The development of the model was guided by the hypothesis, that the given design problem is to be solved by realizing a model that is able to successfully reconstruct the performance attributes and the panel itself, while maintaining a regularized latent space. Although the design error did show some improvements over the one-hundred epoch training session of the first run of the final model, it is safe to say that the hypothesis did not hold. The design error figures A.6 of the tuned model clearly demonstrate an indifference to the requested attributes. At the same time, figure A.5 shows that the majority of the estimated attributes $\hat{\boldsymbol{x}}_{\boldsymbol{rec}}$ of reconstructed panels are similar to the original attributes $\boldsymbol{x}$, creating a counter-intuitive contrast. The first assumptions were that the model is simply not accurate enough and that the reconstruction needs to be improved, but after more reflection it was clear that there are other mechanics at work.
A latent space that is not decoupling from the performance attributes simply by concatenating the conditionality, as initially assumed and proposed in literature, explains the observed behavior. For other applications, a simple concatenation is sufficient to achieve a decoupled latent space, as the conditionality provides additional value to the decoding of the latent vector. For the performance

attributes in this thesis, it seems like the relation to the panels is too abstract for the model to grasp by itself. It is noteworthy that the observed improvement of the design error in the first training run indicates the start of a decoupling process. But it was slow and time-consuming, therefore a new and efficient method was needed that encourages and accelerates the decoupling. With the assumption that the attributes are part of the encoding in the latent space, the model is able to reconstruct the panels without a need for the conditionality, since the distribution of the latent vector for the reconstruction of a specific panel $\boldsymbol{z_i}$ has much lower variance than a normal Gaussian. Therefore, it is not encouraged to learn the propositions of the complex conditionality since clearer information is available in the latent space. However, the latent space should not suggest performance attributes in any form, rather the decoder $\boldsymbol{p(w|x,z)}$ should interpret the latent vector $\boldsymbol{z}$ based on the given conditionality and adjust the generative behavior. In fact, for each possible combination of attributes, a unique decoder $\boldsymbol{p(w|z)}$ is embedded within the general decoder model $\boldsymbol{p(w|x,z)}$. The tuned model seems to only have a single and constant decoder $\boldsymbol{p(w|z)}$. The role of the introduced design loss is to decouple the latent space from the conditionality, by penalizing a decoder of the form $\boldsymbol{p(w|z)}$. If a *fft mean* attribute of 0.1 is requested, but the random sampling of the latent space suggests a panel with an estimated *fft mean* mean value of 0.9 and the model is penalized for the difference, the decoder discovers the underlying meaning of the conditionality rather quickly. Eventually, it is able to reorganize the encoding and decoding of the latent space, removing all redundant information that is already given, and decouples the latent space from the conditionality.

5.3 The Dilemma of the Design Error

From the beginning, it was clear that introducing a design error into the loss function can result in unpredictable behavior. This was exactly what happened in the first session of the design error tuning; the design loss was rapidly improving, while the validation x loss started to sky-rocket. Closing the loopholes by freezing sub-networks stabilized the unpredictability and enabled the realization of the final model. However, there are still uncertainties with the achieved results. An interesting observation is that for some areas of figure A.9, the design error is lower than its own measure of uncertainty, the attribute error displayed in figure 4.23. It is important to be aware that even though this is an estimation of the true design error, the model treats this as the absolute error. Considering that the model struggles to reconstruct the attributes of a panel with the property of $x_{fft\_mean} = 1$, it would not be beneficial to generate a hypothetically perfect panel when $x_{fft\_mean} = 1$ is requested, since it would automatically lead to a relatively high error based on its own evaluation. Thus, because the model is intelligent, it exploits its own weakness as it knows what properties its own analysis pipeline expects, which are not necessarily in correspondence with $x_{fft\_mean} = 1$. This is not to say that the model is not able to perform in a practical application. Most likely it would perform even better if it were able to train on a true design error, since the evaluation would be more coherent. But it means that the models perception of reality is approximately 5% inaccurate, which makes the design error hard to interpret.

5.4 Outlook

There is no shortage of possibilities to build on this work or apply the concepts in a different setting. The current model would benefit the most from an improvement of the x reconstruction, which is a challenging endeavor. A significant amount of resources has been invested to get to this point, but it remains the main source of uncertainty. Getting access to the true performance attribute analysis process, in order to validate a small amount of generated panels, would already bring a lot more clarity. Modeling the joint probability distribution of the performance attributes would enable the user to under-define the conditionality, in order to generate more degrees of freedom for a wider exploration of solutions. Also, it allows random sampling from the joint distribution and remove the dependency from the dataset to generate panels with random conditionalities.
However, applying the concept on a new subject with critical importance on the visual aspects of

the design instance would assumingly provide the most value. Acoustic panels are mainly judged on their functionality and subjective aspects are secondary. Therefore, solution exploration has not the highest priority in the design process. Even though a designer of acoustic panels undoubtedly benefits from a generative model as described in this thesis, there is still more untapped potential that can be realized with this concept.

5.5 Conclusion

The concept of reversing the parametric design process proposed by Salamanca et. al. [9] has successfully been implemented for the subject of acoustic panel design. It required the embedding of a cnn within the model architecture and introducing variationality into the system. These architectural additions generalize the model in the presented paper and enable the concept to be more inclusive with respect to potential applications. Throughout the course of the thesis, relevant findings have been presented, such as the value of supplementary data representations, an adaption to the classical VAE loss and opening a discourse about potential modifications to the KL divergence. The analysis pipeline of the model has been developed and enabled a self-evaluating feedback loop. A novel method to decouple the latent space of CVAE models has been presented, that makes use of the feedback loop in form of a design error loss. The investigation and eventual disproving of the initially posed hypothesis resulted in a better understanding of the dynamics of latent spaces in CVAE models. Lastly, the final model and the intended use case of design exploration is validated by guiding the reader through an application of design exploration conditioned on requested performance attributes.

Appendix A

Results

A.1 Models

A.1.1 Base Model

Reconstruction Error

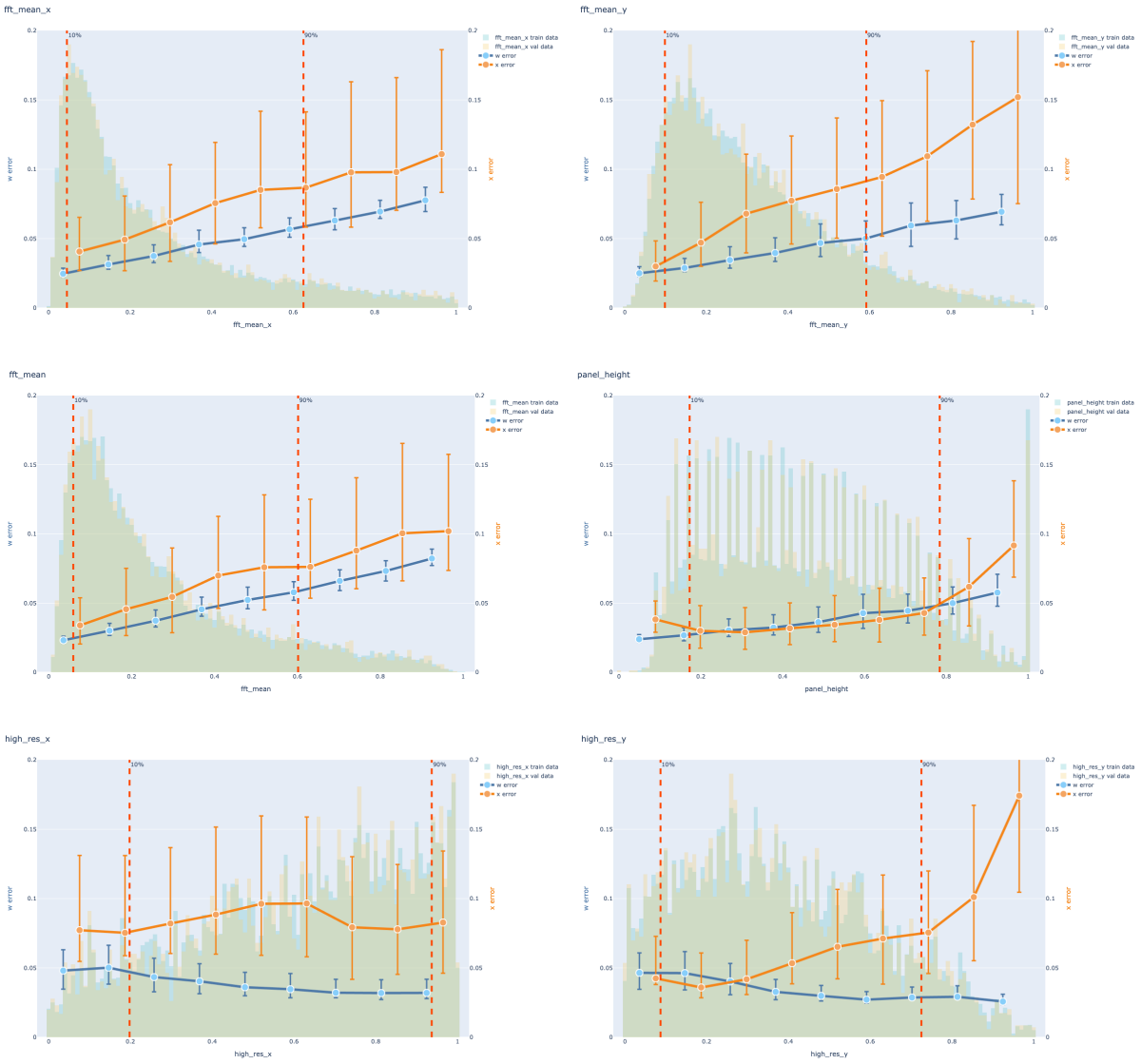


Figure A.1: Error of panel and attribute reconstruction on validation data.

Qualitative Reconstruction

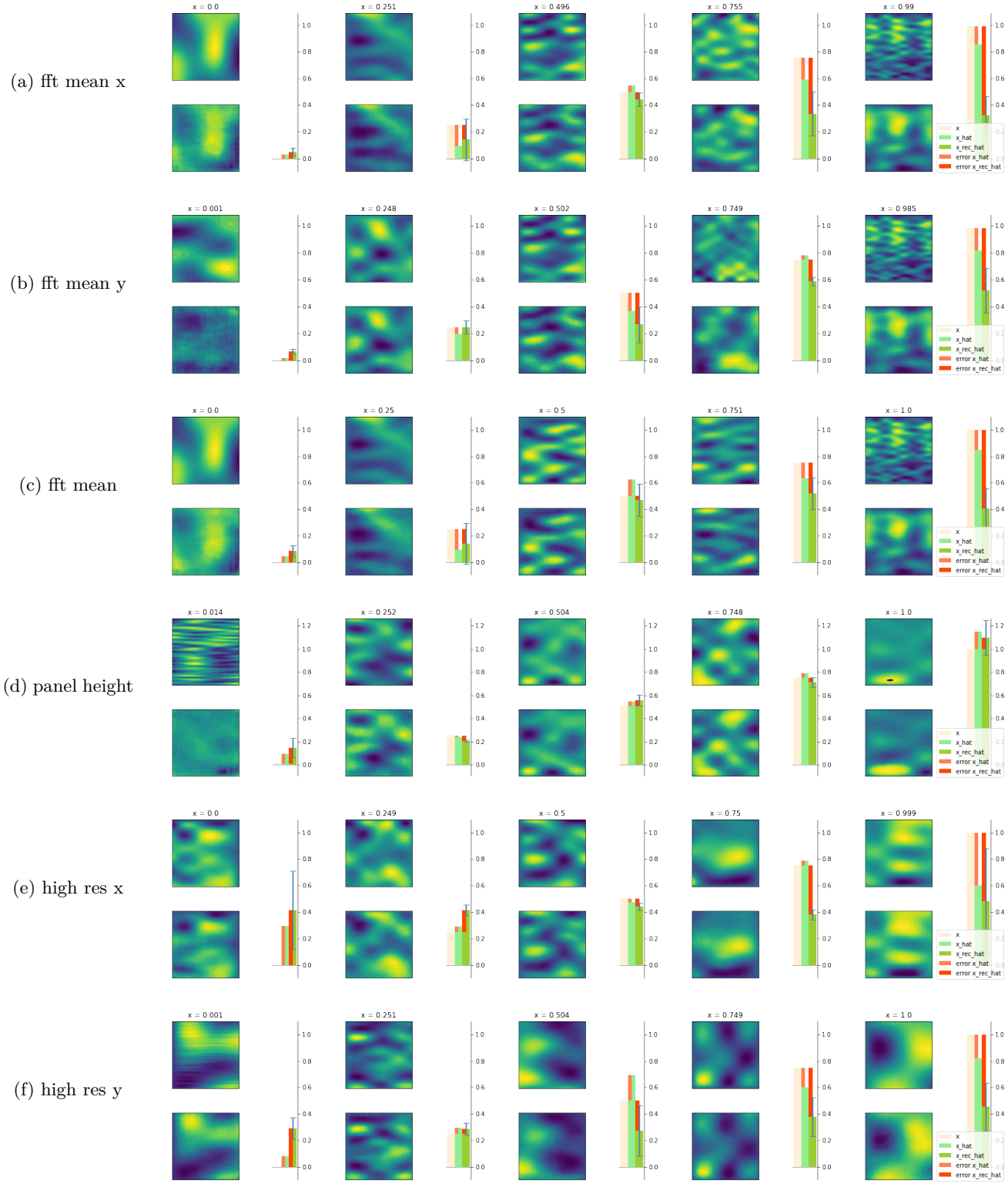


Figure A.2: Reconstruction of panel and performance attribute on validation data.

Design Error

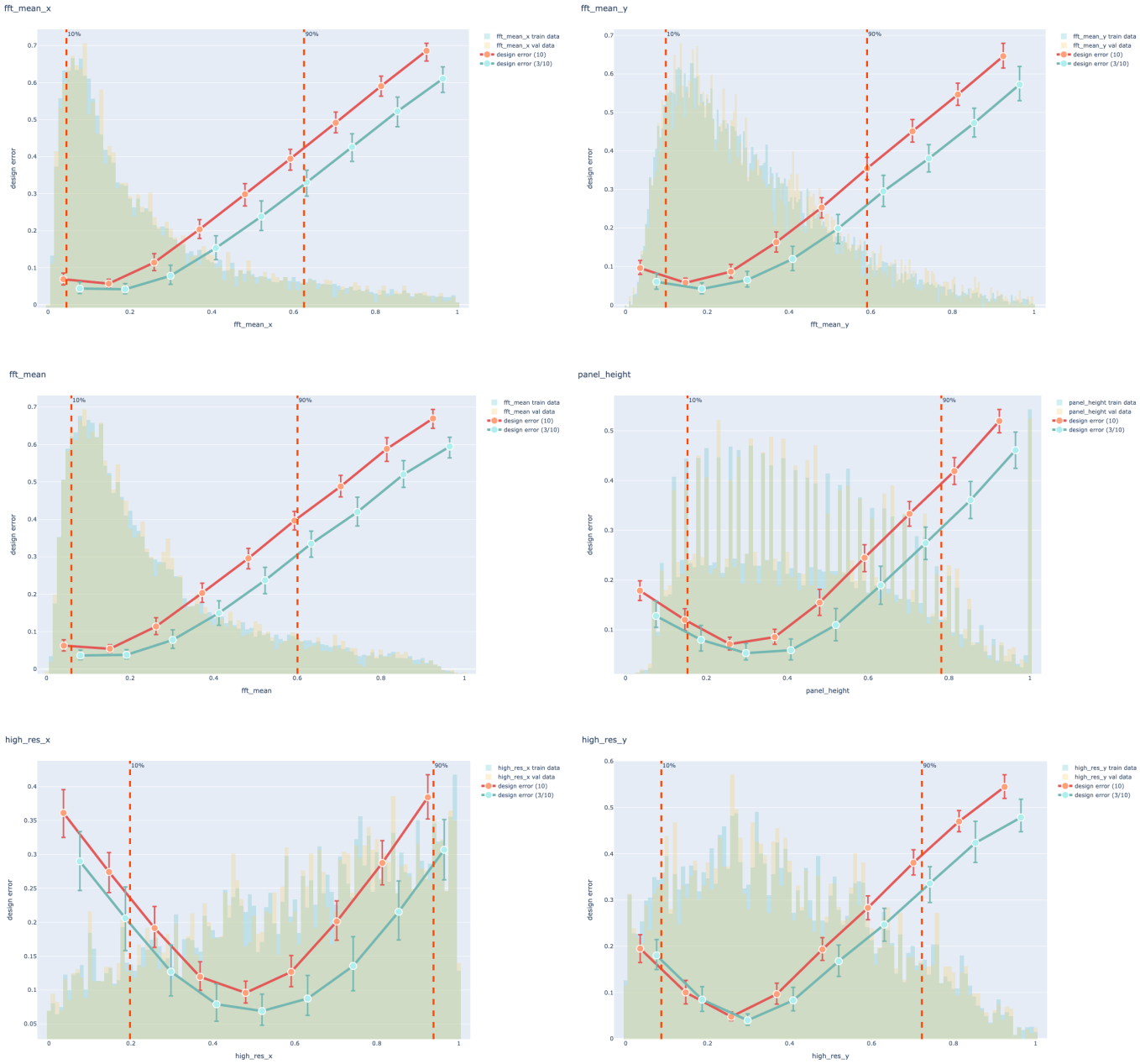


Figure A.3: Design errors of requested attribute on validation data.

A.1.2 Tuned Model

Reconstruction Error

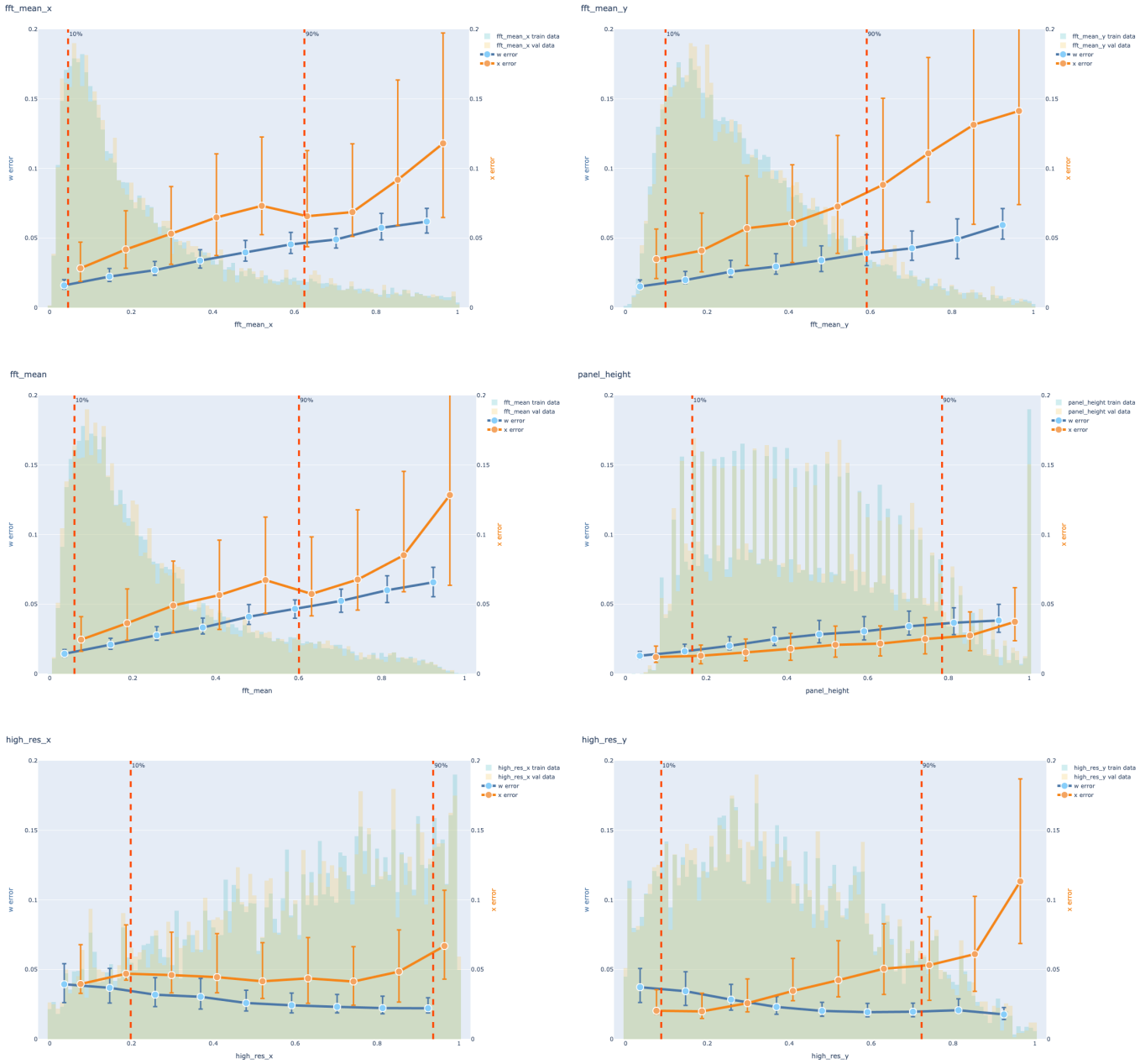


Figure A.4: Error of panel and attribute reconstruction on validation data.

Qualitative Reconstruction

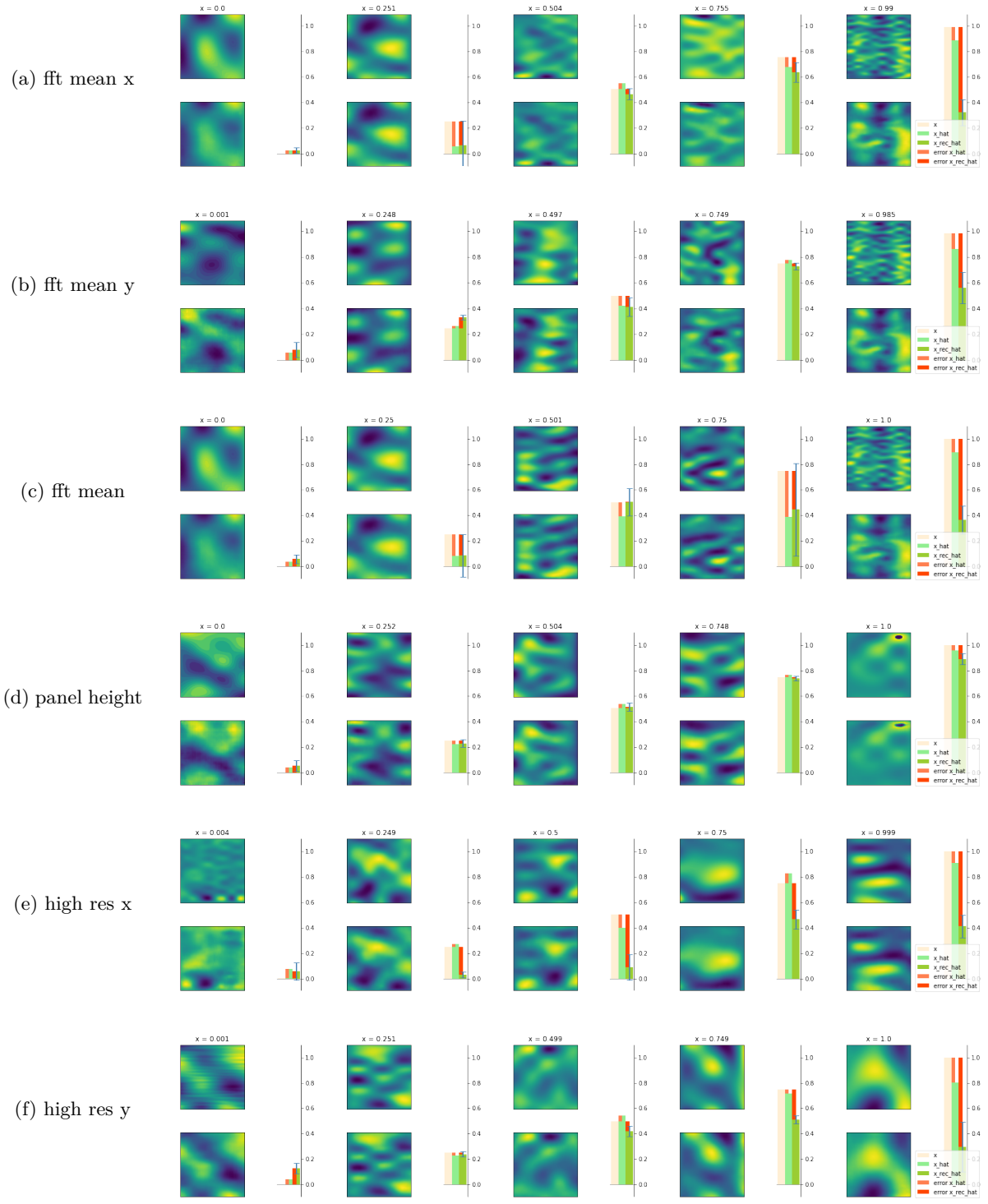


Figure A.5: Reconstruction of panel and performance attribute on validation data.

Design Error

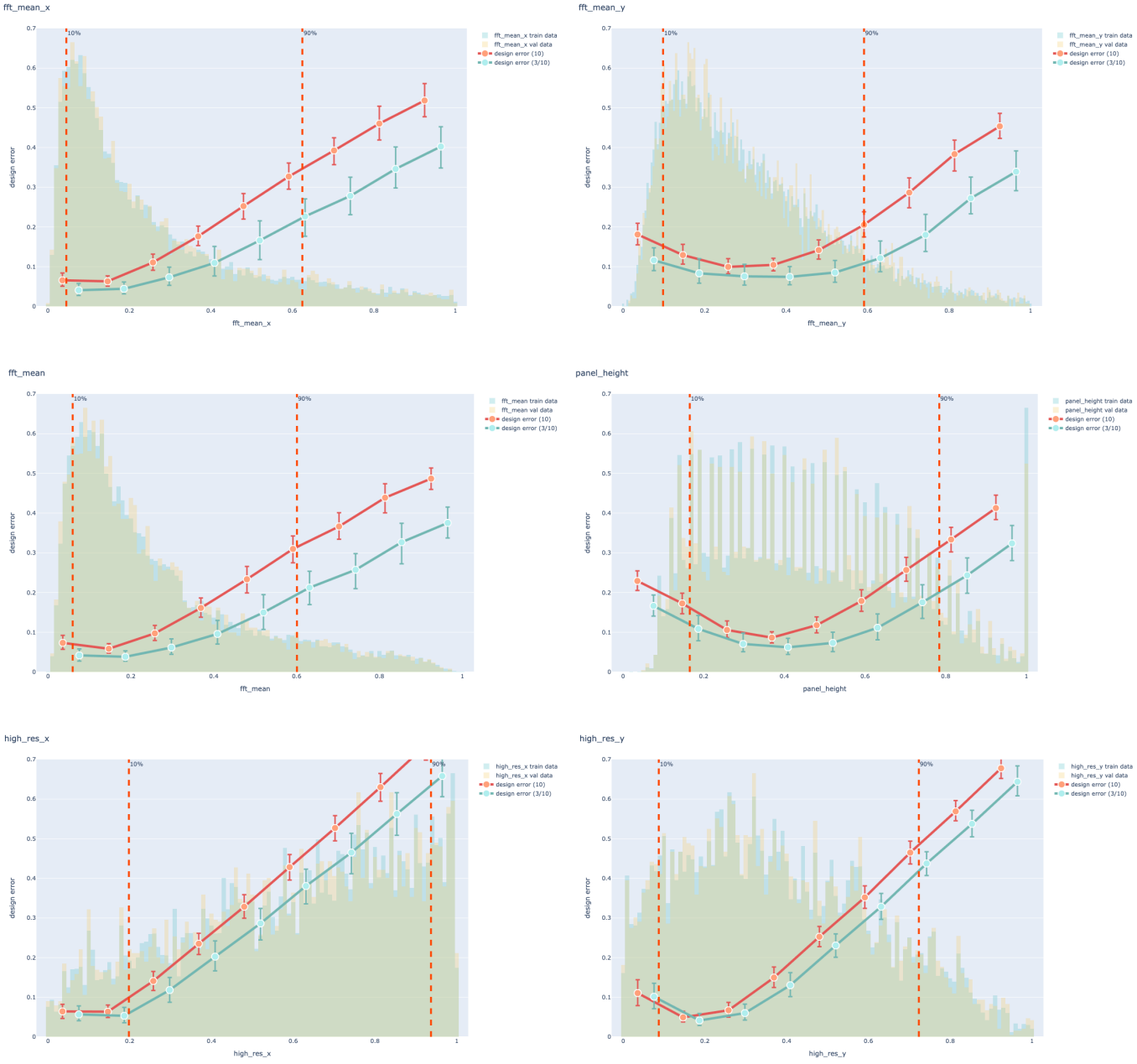


Figure A.6: Design errors of requested attribute from validation data.

A.1.3 Final Model

Reconstruction Error

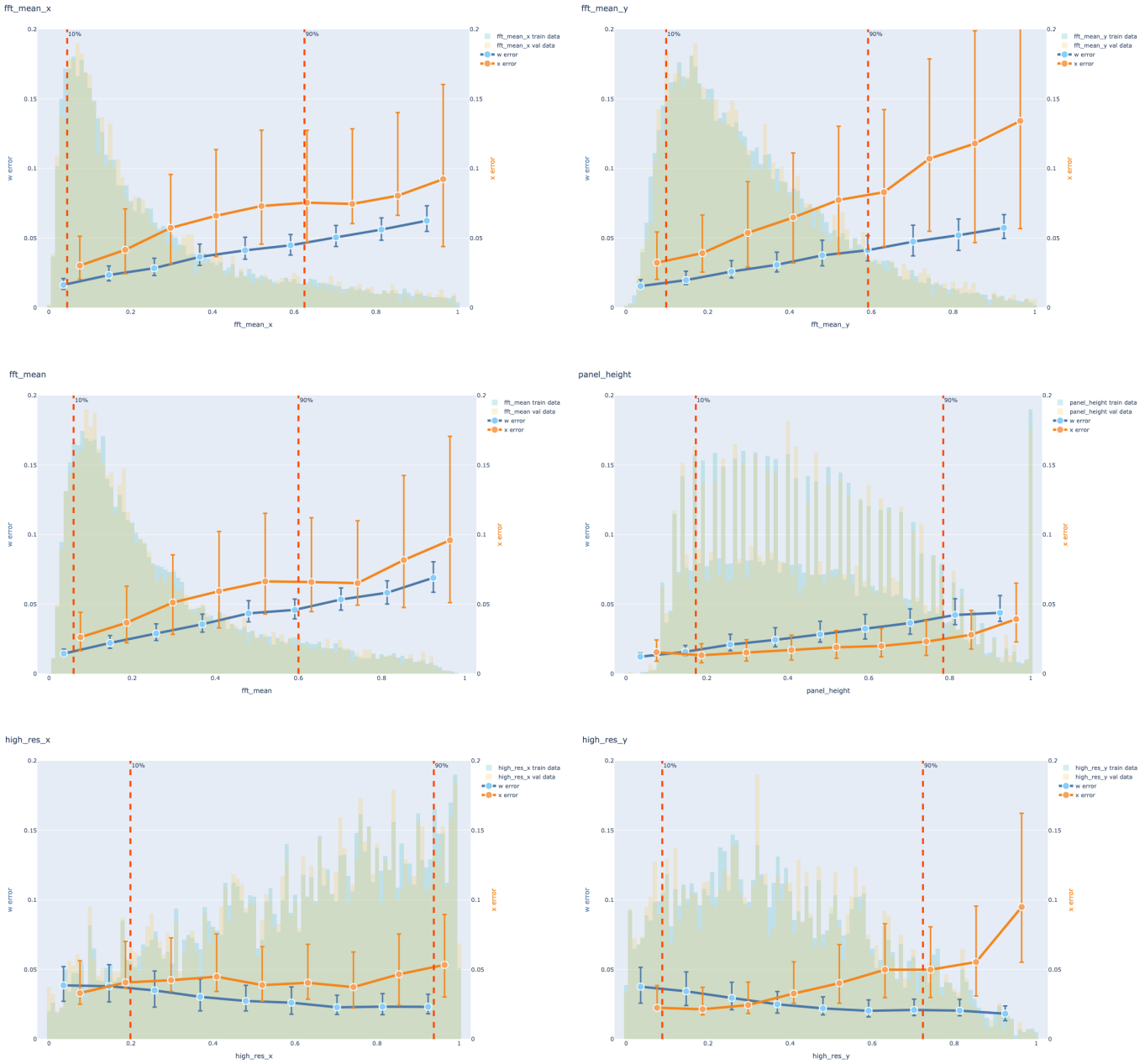
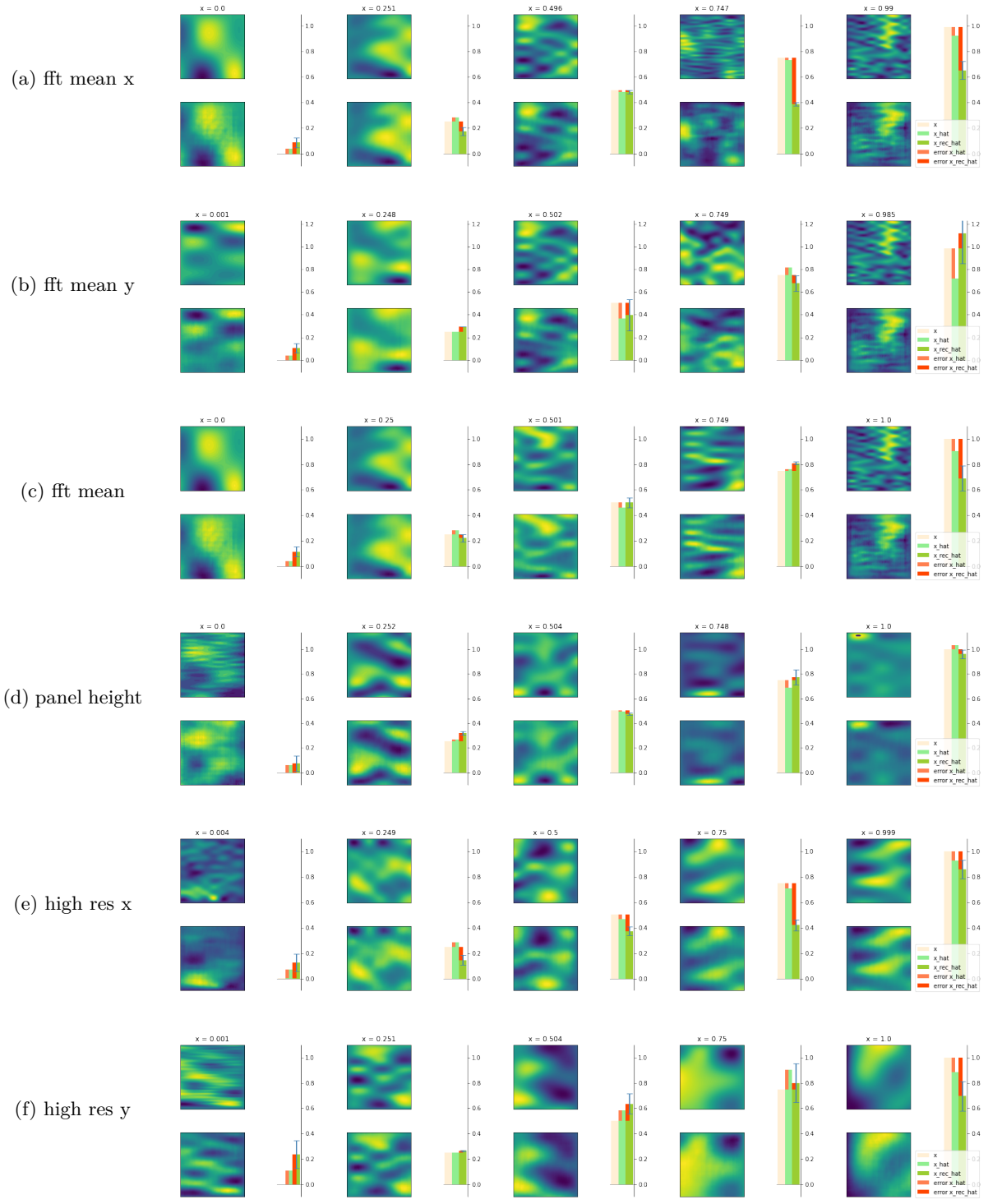


Figure A.7: Error of panel and attribute reconstruction on validation data.

Qualitative Reconstruction



(a) fft mean x

(b) fft mean y

(c) fft mean

(d) panel height

(e) high res x

(f) high res y

Figure A.8: Reconstruction of panel and performance attribute on validation data.

Design Error

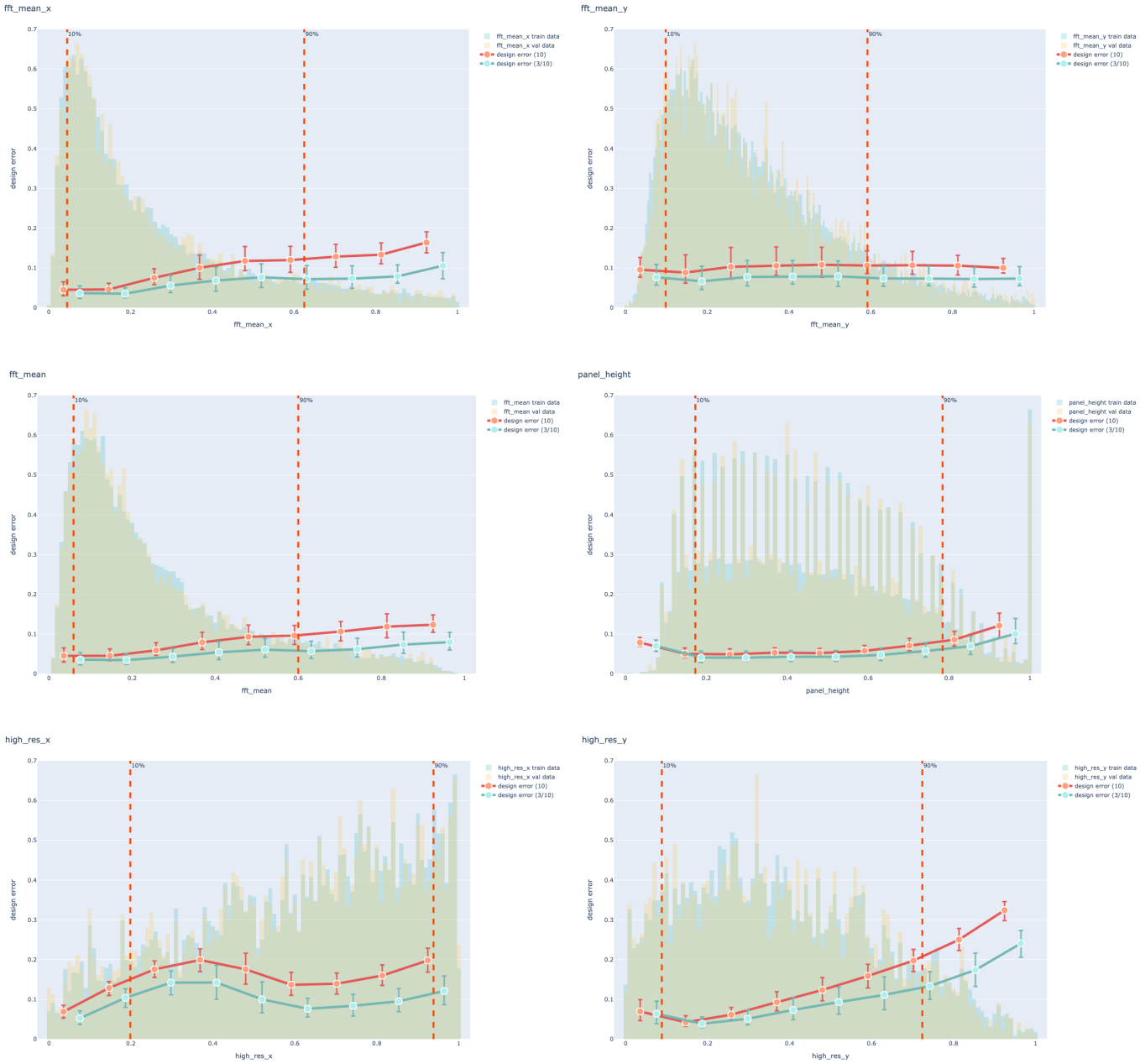
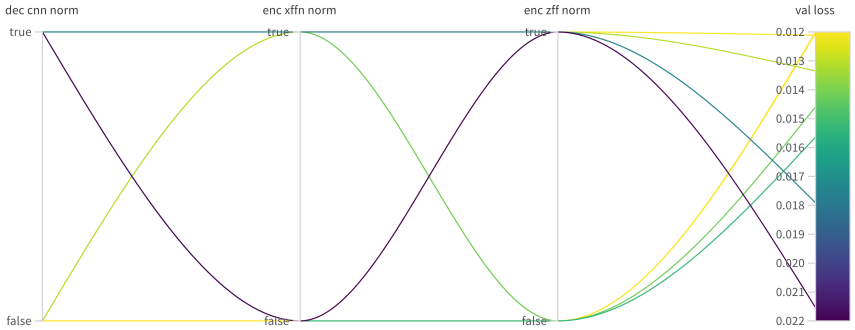


Figure A.9: Design errors of requested attribute from validation data.

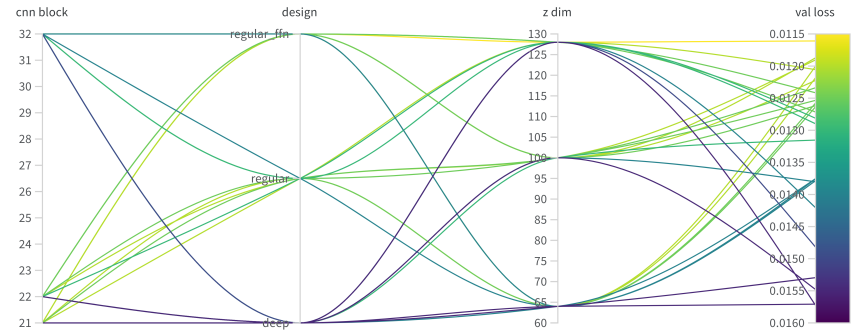
A.2 Hyperparameter Tuning

A.2.1 Tuning Set 1

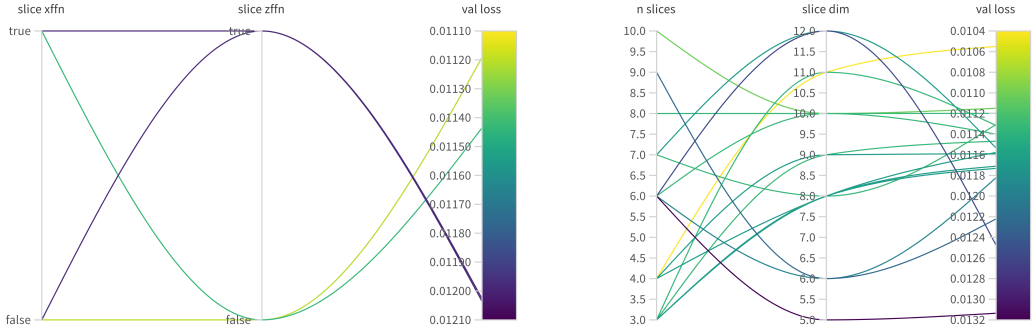
(a) Batch norm



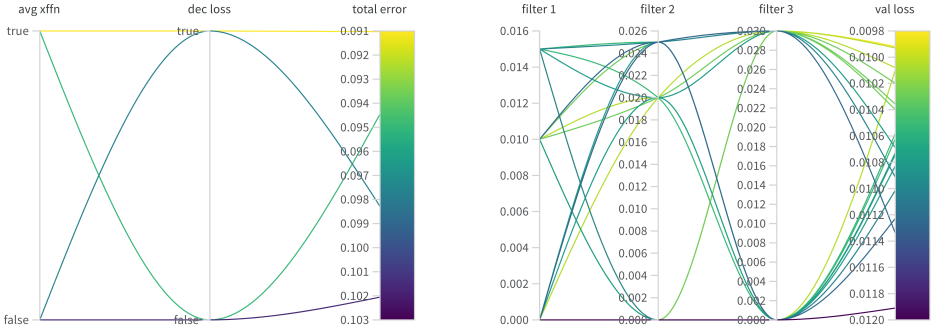
(b) Net design



(c) Slicer

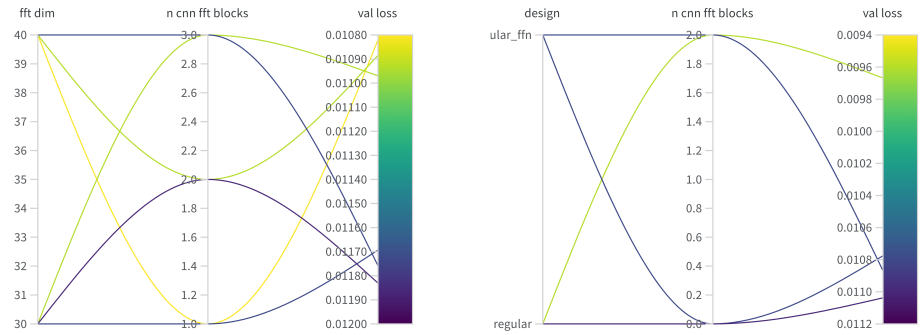


(d) FFT filter

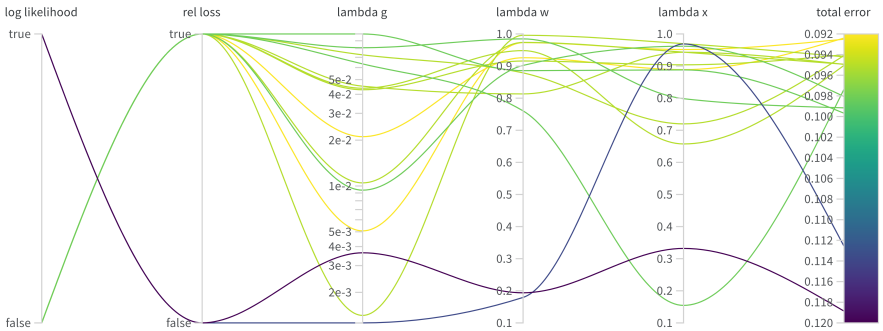


A.2.2 Tuning Set 2

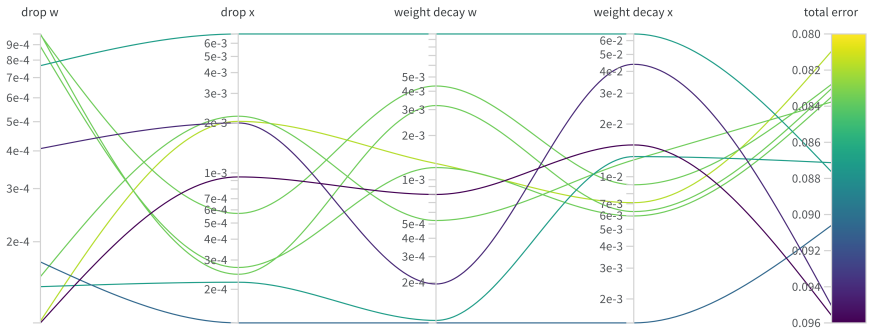
(a) FFT cnn



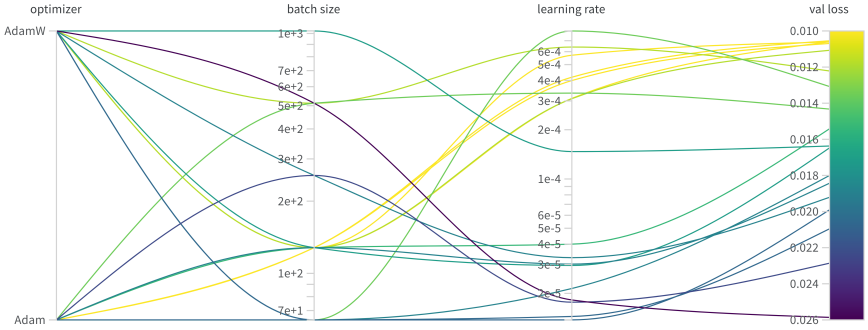
(b) Reconstruction loss



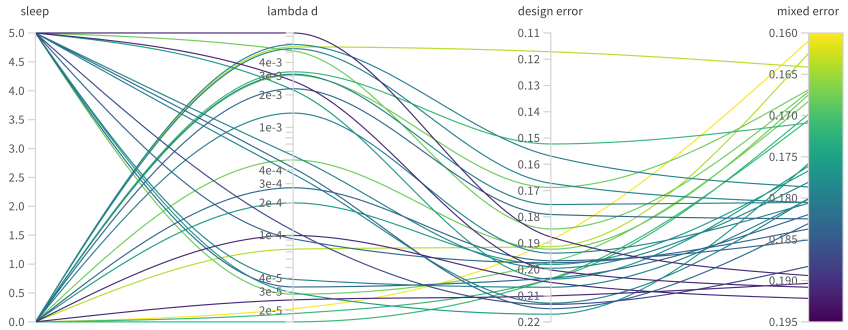
(c) Dropout & weight decay



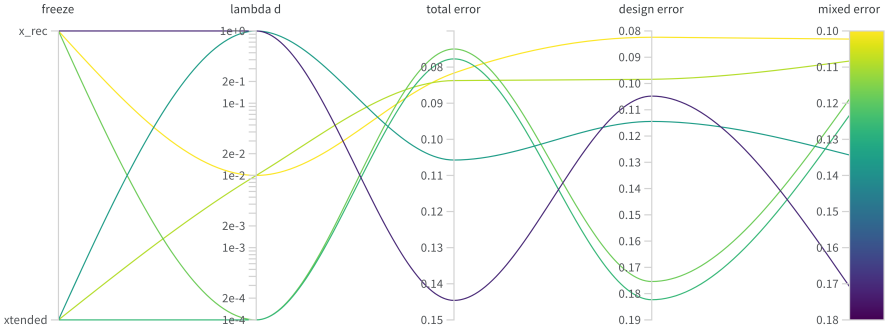
(d) Optimizer



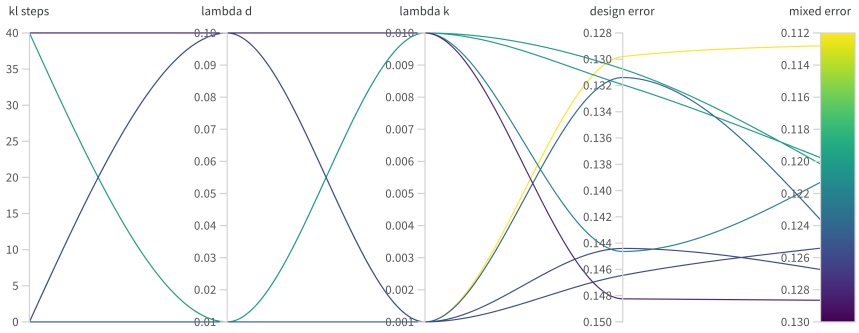
A.2.3 Design Error Tuning



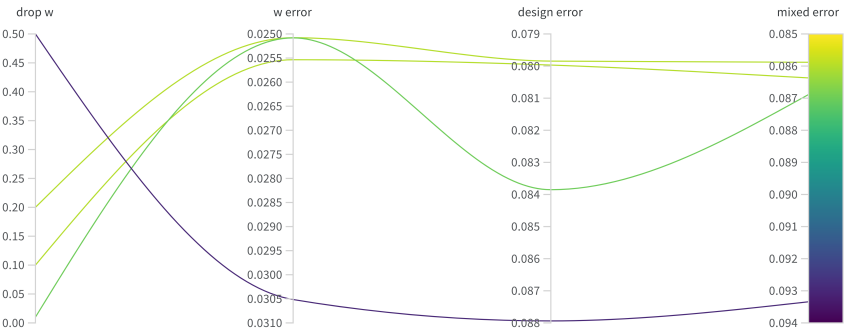
(a) Introduce lambda d



(b) Freeze sub-networks



(c) Design and kl loss



(d) Dropout

Figure A.12: Overview of design error tuning sessions.

A.3 Datasets

A.3.1 Original Dataset

Attribute Density

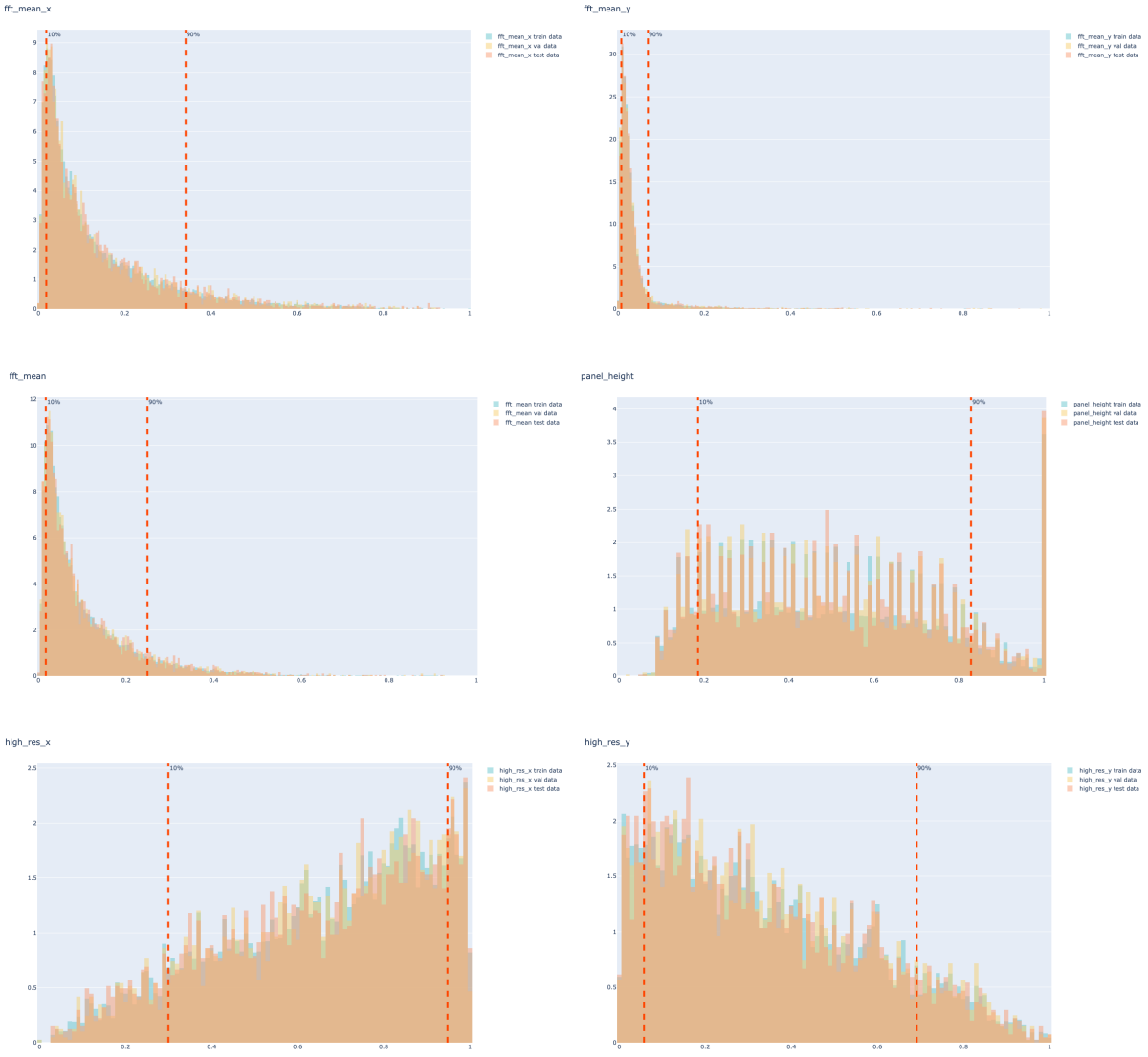
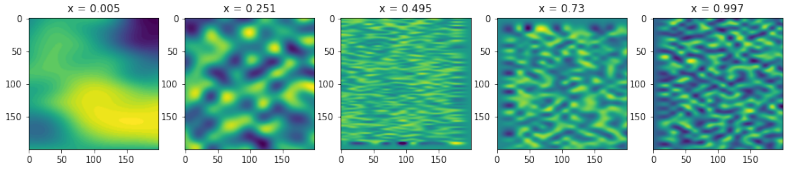


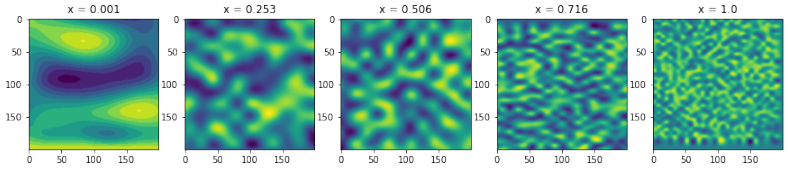
Figure A.13: Density of performance attributes on original dataset.

Panels

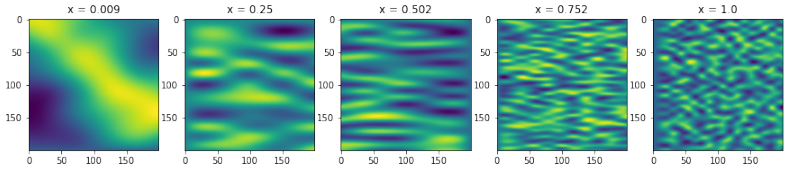
(a) fft mean x



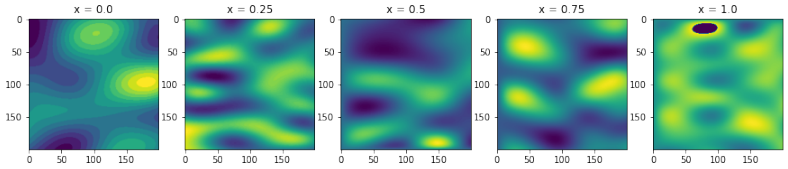
(b) fft mean y



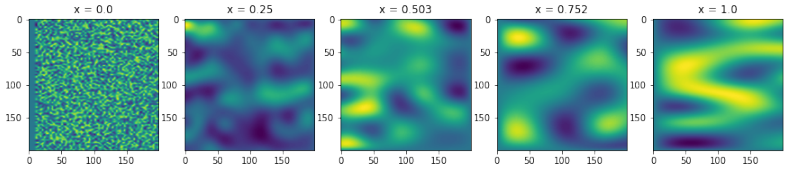
(c) fft mean



(d) panel height



(e) high res x



(f) high res y

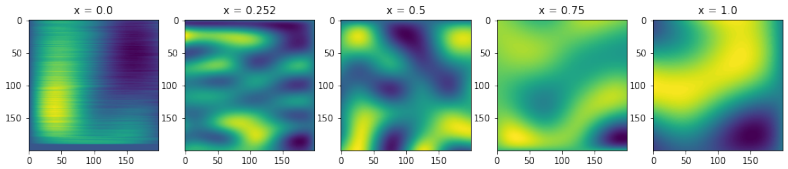


Figure A.14: Range of original dataset for each attribute.

A.3.2 Final Dataset

Attribute Density

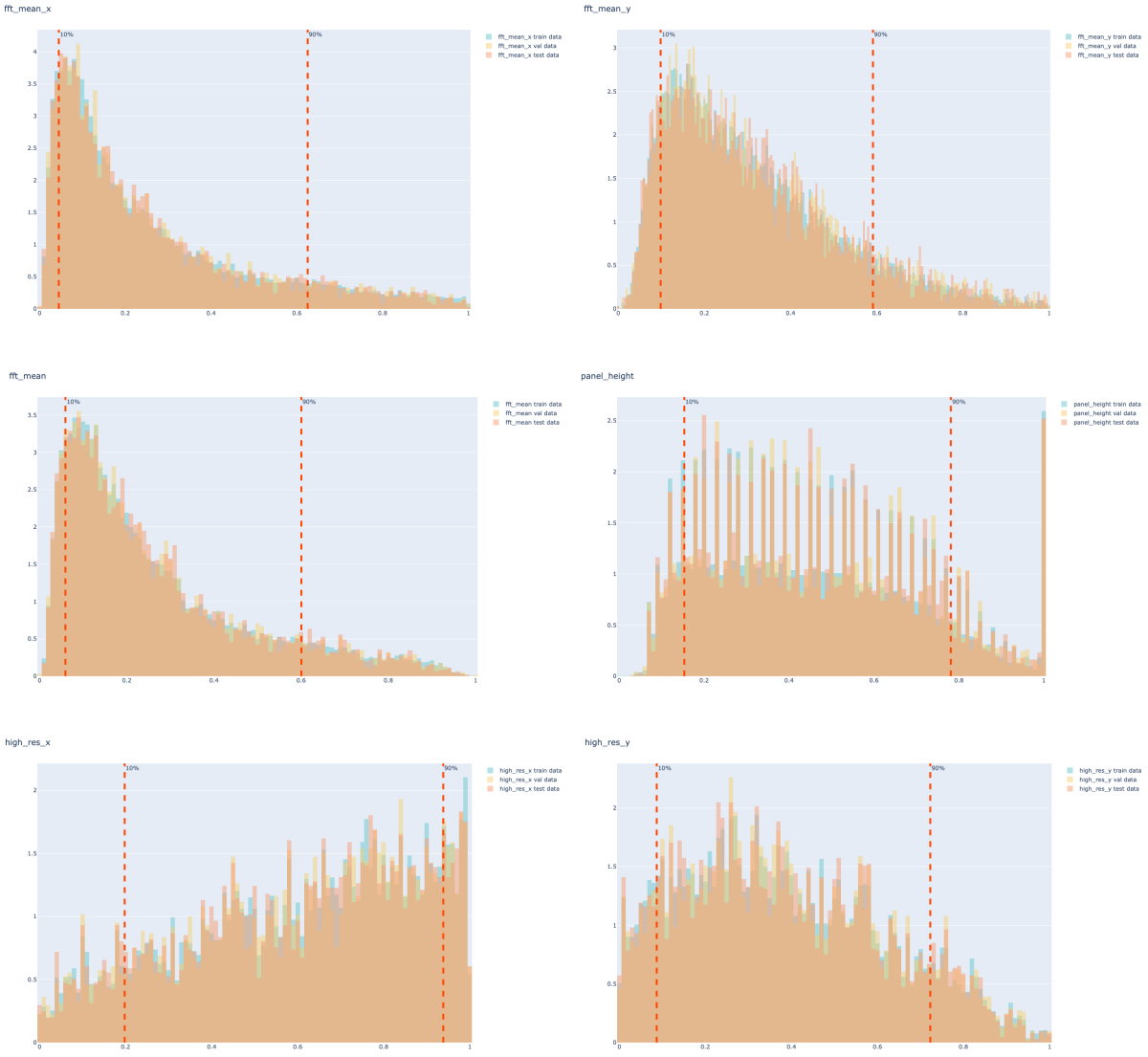


Figure A.15: Density of performance attributes on final dataset.

Panels

(a) fft mean x

(b) fft mean y

(c) fft mean

(d) panel height

(e) high res x

(f) high res y

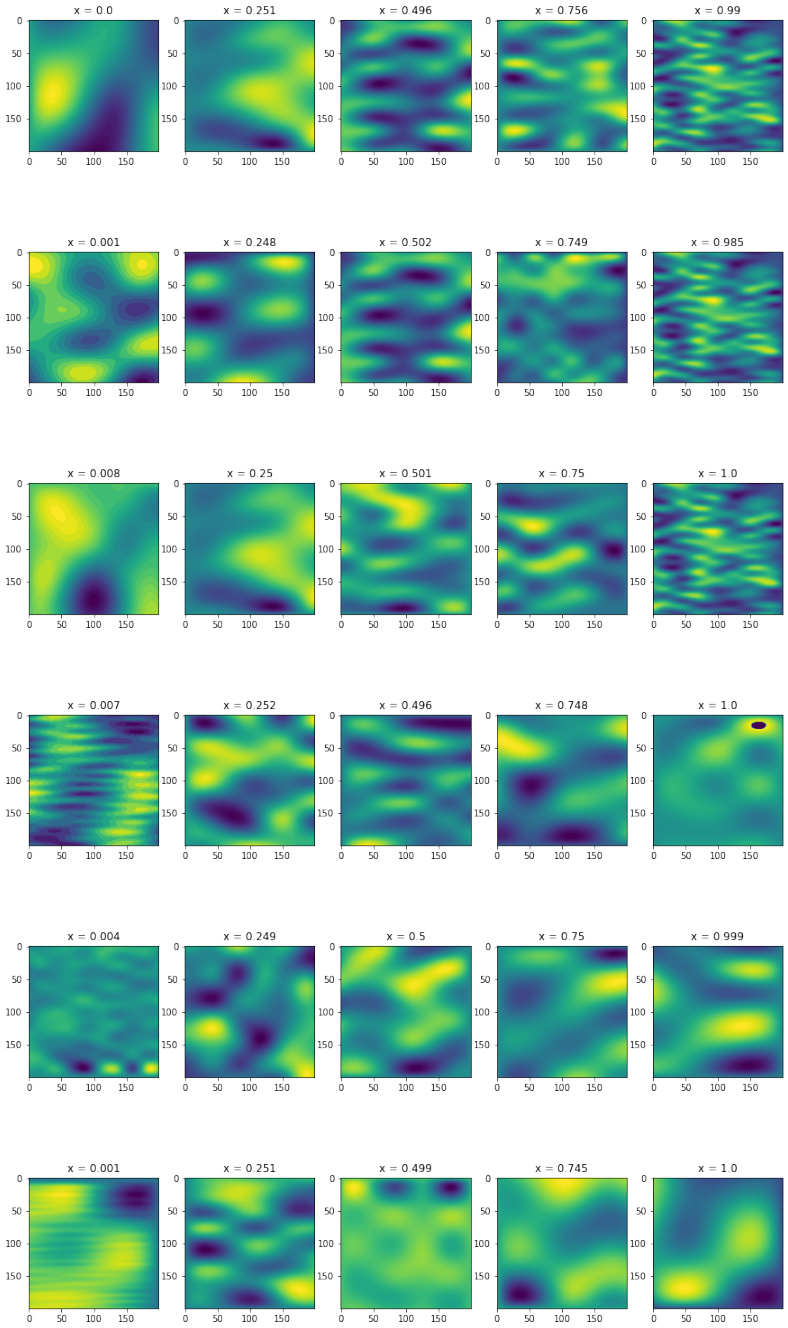


Figure A.16: Range of final dataset for each attribute.

A.4 Code Repository

The link to the code repository:
panels-cvae-repo

Please refer to the included README file for any further instructions.

Bibliography

[1] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[2] Jianmin Bao, Dong Chen, Fang Wen, Houqiang Li, and Gang Hua. Cvae-gan: fine-grained image generation through asymmetric training. In *Proceedings of the IEEE international conference on computer vision*, pages 2745–2754, 2017.

[3] E Oran Brigham. *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

[4] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[5] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23(2):358–367, 1988.

[6] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[9] Matthias Kohler Luis Salamanca, Aleksandra Anna Apolinarska and Fernando PÃ©rez-Cruz. *Augmented Intelligence for Architectural Design with Conditional Autoencoders: Semiramis case study*.

[10] Sangeun Oh, Yongsu Jung, Seongsin Kim, Ikjin Lee, and Namwoo Kang. Deep generative design: Integration of topology optimization and generative models. *Journal of Mechanical Design*, 141(11), 2019.

[11] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[12] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[13] Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, 33:19667–19679, 2020.

[14] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Thesis template provided by:
Engineering Design and Computing Laboratory
Prof. Dr. K. Shea

Title of work:

Design Exploration of Acoustic Panels with Conditional Variational Autoencoders

Thesis type and date:

Master Thesis, July 2022

Supervision:

Prof. Dr. Mirko Meboldt
Prof. Dr. Fernando Perez-Cruz
Dr. Luis Salamanca

Student:

Name:	Luca Bettermann
E-mail:	lucabe@student.ethz.ch
Legi-Nr.:	15-918-394
Semester:	6

Statement regarding plagiarism:

By signing this statement, I affirm that I have read and signed the Declaration of Originality, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Declaration of Originality:

`http://www.ethz.ch/faculty/exams/plagiarism/confirmation_en.pdf`

Zurich, 18. 7. 2022: _____