Faculté sciences et sciences de l'ingénieur

Year 2024

Master 2 - Ingénieurie Mathématique

Research Internship Report

Inferring perceptual scales for textures: a deep learning approach

Student	Félix Watine felix.watine@edhec.com
Company	MAP5, applied mathematics laboratory of Université Paris Cité (Image department) 45 rue des Saints-Pères, 75270 Paris, France
Supervisors	Jonathan Vacher, Maître de Conférence Pascal Mamassian, Directeur de recherche
Reference Professor	Jacques Froment, Professeur des universités
Date of Defense	27 August 2024

Université Bretagne-Sud – UFR Sciences et Sciences de l'ingénieur
Département Mathématique, Informatique, Statistique
Rue Yves Mainguy 56000 Vannes France

www.univ-ubs.fr

Acknowledgments

I extend my sincere gratitude to Prof. Jonathan Vacher for his exceptional guidance and support throughout this research internship. His feedback and encouragement were invaluable.

I also appreciate the MAP5 colleagues for their support and the stimulating atmosphere that made this experience truly enriching.

Abstract and Keywords

This report presents a study on visual perception, focusing on perceptual scales and texture generation. The first part examines the Maximum Likelihood Difference Scaling (MLDS) method, questioning its underlying assumptions and proposing empirical tests to validate its accuracy. The second part explores texture synthesis using Variational Autoencoders (VAEs), addressing challenges such as blurriness and high dimensionality through architectural modifications and advanced techniques. The findings highlight the need for refined loss metrics and suggest potential improvements using modern generative models.

Keywords: Visual perception, perceptual scales, psychometric function, 2AFC, Maximum Likelihood Difference Scaling (MLDS), texture synthesis, texture interpolation, Variational Autoencoder (VAE), deep learning, generative models, planar flow, Generative Adversarial Network (GAN), multiscale pyramid, sparsity, kernel

Contents

1 Introduction

The internship takes place at MAP5, an applied mathematics laboratory affiliated with the Université Paris Cité (formerly Université Paris Descartes) and the CNRS. My supervisors, Jonathan Vacher (a professor at MAP5) and Pascal Mamassian (a CNRS research director at the École Normale Supérieure and director of the Laboratoire des Systèmes Perceptifs (LSP)) are both conducting research in the field of visual perception.

1.1 Visual Perception

Visual perception can be defined as the ability to detect and process light such as to interpret and make sense of visual information. The visual system is the physiological basis of visual perception. The visual system includes the eyes, the connecting pathways through to the visual cortex and other parts of the brain.

The eye is a sophisticated biological image acquisition system shaped by billions of years of evolution. When light enters the eye, it passes through the cornea and the lens, which act as the primary focusing elements by refracting light onto the retina. The pupil, a hole in the center of the iris, controls the amount of light entering the lens by dilating or constricting in response to changing light conditions to optimize vision. The retina contains specialized photoreceptor cells called rods and cones, which convert light into electrical signals (visual phototransduction) that are then transmitted to the brain via the optic nerve.

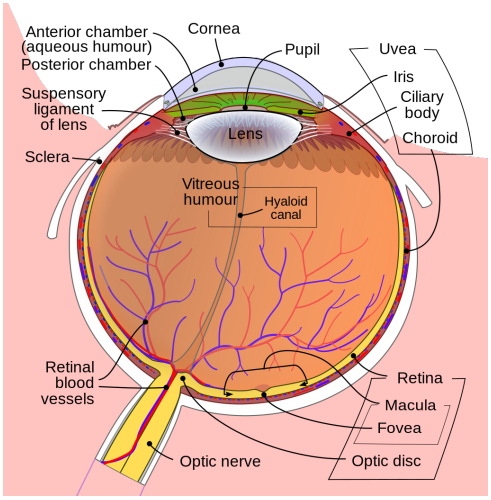

Figure 1: The human eye

The visual cortex, situated in the occipital lobe of the brain, processes the visual information, in form of electrical signals, received from the optic nerve. The primary visual cortex (V1) is the initial processing area. It plays a vital role in edge detection and spatial processing, with neurons tuned to specific orientations and spatial frequencies. As visual processing progresses, information flows into higher visual areas like V2, V3, and V4, where more complex features, including texture, depth, and color, are analyzed. These areas build upon the basic visual attributes detected in V1, contributing to the formation of a detailed internal representation of the visual scene. Additionally, regions like V5 and V6 play essential roles in motion perception and the integration of visual information with other sensory inputs, contributing to our comprehensive understanding of the surrounding environment. Thus, the visual cortex operates as a hierarchical network, with each area contributing uniquely to the construction of our internal representation.

Understanding visual perception is vital because it forms the cornerstone of how we interact with the world around us. Our ability to perceive and interpret visual information influences every aspect of our lives. In practical terms, visual perception guides our actions and movements, allowing us to navigate safely through our environment by recognizing obstacles and hazards. Moreover, it facilitates communication and social interaction by

enabling us to interpret facial expressions, gestures, and body language. Additionally, visual perception aids in the acquisition of knowledge, comprehension of concepts, and retention of information, particularly in educational settings where visual aids are commonly used. It allows us to understand complex ideas presented in diagrams, charts, and illustrations, enhancing our learning experience.

The study of visual perception lies at the intersection of neuroscience, psychology, and applied mathematics. Understanding the visual system involves examining both brain signals and the subjective sensations generated by visual stimuli. This requires not only the modeling of these signals and sensations but also the precise mathematical definition of visual stimuli. Modeling and understanding visual perception is challenging due to the high dimensionality and variability of visual inputs, such as natural images. The subject of my research can be divided into two main parts:

1. Exploring theoretical aspects of visual perception, with a particular focus on the Maximum Likelihood Difference Scaling (MLDS) method.

2. Developing methods for generating textures using deep learning techniques, which will be utilized in experiments related to visual perception.

1.2 Maximum Likelihood Difference Scaling (MLDS)

In visual perception, one key concept is the existence of a perceptual scale. It describes the relationship between the physical magnitudes of a stimulus on the one hand and the internal perception of a stimulus on the other hand. The perceptual scale can be represented by a function $\phi : \mathcal{S} \subseteq \mathbb{R} \to \mathcal{R} \subseteq \mathbb{R}$, where a physical stimuli s is transformed by a (in general non-linear) function into a psychological representation $\phi(s)$. Examples of perceptual scales include the relationships between perceived contrast and physical contrast, perceived depth and physical depth, or perceived velocity and physical velocity. The Maximum Likelihood Difference Scaling (MLDS) method, developed by Maloney and Yang [9], measures this perceptual scale by comparing the perceived differences between two pairs of stimuli.

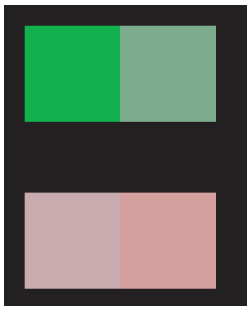

Figure 2: Two pairs

As an example, consider the pairings of color samples [1] depicted in Figure 2. Observers can distinguish between the colors in each pair and most of them when asked to select the pair with the greater color difference, would choose the upper pair.

The scenario presented in Figure 2 exemplifies a typical trial in a MLDS experiment. Here, the observer assesses two pairs of stimuli, constituting a quadruple, and identifies the pair with the larger perceived difference. All four color samples in Figure 2 are positioned along a line in color space, illustrated in Figure 3. Specifically, the upper pair in Figure 2 corresponds to the 1st and 5th color samples in Figure 3, while the lower pair corresponds to the 7th and 8th samples. Throughout the experiment, the observer repeats this judgment for numerous quadruples of color samples, all drawn from Figure 3.

[1] example taken from [9]

The objective of MLDS is to assign numerical values $\phi(s_1), \phi(s_2), \ldots, \phi(s_{11})$ to the 11 stimuli in Figure 3, such that the absolute differences between these scale values accurately predict the observer's judgments. For instance, if the observer consistently perceives the difference between the upper pair in Figure 3 as greater than that of the lower pair, then regardless of the specific values chosen for $\phi(s_1), \phi(s_2), \ldots, \phi(s_{11})$, they should satisfy $|\phi(s_2) - \phi(s_5)| > |\phi(s_7) - \phi(s_8)|$. MLDS utilizes responses from a large number of quadruples. It then assigns scale values that best align with the experimental data.

Figure 3: Color scale

1.3 Texture Synthesis

In visual perception studies, researchers often prefer artificial, well-controlled stimuli over complex natural images. One type of image which is of interests are textures. A minimal definition of a texture is an "image containing repetitive patterns." There is no broadly accepted mathematical definition of a texture. Depending on the nature of the texture, the pattern family can be more or less random. According to a probabilistic hypothesis, a texture is the realization of a stationary random field. A random field $\{X(s) : s \in \mathbb{R}^2\}$ is said to be stationary if its statistical properties are invariant under translations in the plane. Specifically, for any $s_1, s_2, \ldots, s_n \in \mathbb{R}^2$ and any $\tau \in \mathbb{R}^2$, the joint distribution of $(X(s_1), X(s_2), \ldots, X(s_n))$ is the same as that of $(X(s_1 + \tau), X(s_2 + \tau), \ldots, X(s_n + \tau))$. This stationary property is visually evident in the texture shown in Figure 4.

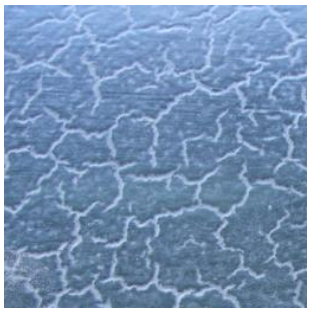

Figure 4: Texture

In natural images, textures have various origins such as materials (wood, stone, fabric, sand), vegetation (grass, moss, foliage), and natural phenomena (water surface, clouds). Generally, two main sub-families are considered: micro-textures and macro-textures, the latter being composed of small but individually discernible objects. Texture synthesis, especially the controlled generation of natural textures, provides a valuable tool for studying visual perception. Texture synthesis algorithms can be divided into two main classes: synthesis based on statistical constraints and synthesis by patches (also known as "copy-paste" algorithms). In the former class, the first step consists in extracting relevant "statistics" from the input image (e.g., color distribution, Fourier coefficients, wavelet coefficients, etc.). The second step consists in computing a "random" output image that has the same statistics: start with a white noise image and alternately impose each "statistic" from the input image.

A key hypothesis in visual perception is the Bayesian modeling of perception, developped by Knill and Pouget [7]. Humans use their sensory organs to get a perception of the state of the world, but the information provided from the sensory organs is incomplete. A two-dimensional retinal image is compatible with infinitely many three-dimensional objects. Our brains must effectively deal with the resulting uncertainty to generate perceptual representations. To do so, the brain uses prior knowledge to interprete visual perception. For example, the internal representation that we make from the floor based

on the sensory information received is conditioned on the prior knowledge of the weather. In fact, if it has just rained, the "shiny effect" on the floor would be attributed to the wetness, but if it has been dry, the "shiny" effect on the floor would be attributed to the type of surface. The Bayesian modeling of perception is a continuation of the concept of information maximization, i.e the brain maximizes the mutual information between stimulus and response [9]. The Bayesian model supports the texture synthesis approach based on statistics, as it supports the fact that visual perception is probabilistic. In fact, a hypothesis often made is that textures with similar statistics are indistinguishable to humans (Julesz hypothesis) [6].

One topic of interest in visual perception is the understanding of perceptual scale between textures. In a similar fashion that we defined a color scale in Figure 3, we want to define a texture scale (ex: from water surface to clouds). However, there is no natural physical parameter that continuously links two textures. As we will see, deep learning can help.

1.4 Problem Addressed

1.4.1 Testing the hypothetis behind MLDS

The observer's model behind MLDS One of the objectives of my internship is to question the hypothesis that is used when performing MLDS. In fact, MLDS assumes that an observer, when being forced to pick one of the two pairs (mathematically it means deciding whether $\Delta_{i,j,k,l} > 0$ or $\Delta_{i,j,k,l} < 0$) is only subject to the following Gaussian noise N (H1):

$$\Delta_{i,j,k,l} = |\phi(s_i) - \phi(s_j)| - |\phi(s_k) - \phi(s_l)| + N \tag{1}$$

where $N \sim \mathcal{N}(0, \sigma)$, s_i, s_j, s_k, s_l represent the stimulis, ϕ is deterministic. Here, the noise N, called additive Gaussian noise, does not depend on the magnitudes of the stimuli or the differences of stimuli under comparison. It is independent and identically-distributed. In practice, MLDS solves for the perceptual scale ϕ by maximum likelihood and assumes the absolute values can be removed, since removing them corresponds to swapping the order of two stimulis within a pair. In fact whether a stimuli s_i is located left or right of s_j has no physical incidence on the experience. Therefore the equation 1 becomes:

$$\Delta_{i,j,k,l} = \phi(s_i) - \phi(s_j) - \phi(s_k) + \phi(s_l) + N \tag{2}$$

which is easier to optimize. This assumption that the noise is external to the stimuli of the experience can be questioned.

The model of the perception of a stimulus Another assumption is that an observer, when observing a single stimuli is subject to noise as described by this equation of perception (H2):

$$M(s_i) = \phi(s_i) + N_i \tag{3}$$

where $N_i \sim \mathcal{N}(0, \sigma)$ Under this assumption, equation 1 can be reformulated as:

$$\widetilde{\Delta}_{i,j,k,l} = |\phi(s_i) + N_i - (\phi(s_j) + N_j)| - |\phi(s_k) + N_k - (\phi(s_l) + N_l)| \tag{4}$$

where N_i, N_j, N_k, N_l are iid. In Section 2, we will analyse these two hyptothesis in depth and show how they lead to different inferring of perceptual scales and how we can test which one is accurate.

1.4.2 Generative models for controlled texture synthesis

We would like to generate textures that we could somehow interpolate in order to create a "texture scale". Note that in visual perception we are interested in how an observer perceives (or not) very small variations in stimuli. So the texture scale should not have holes. To interpolate textures, there exists currently two approaches that both have pros and cons. The first approach is to interpolate between two textures by interpolating their statistics. The Portilla-Simoncelli (PS) algorithm can generate texture from a white noise by iteratively imposing high-order statistical constraints (wavelet transform coefficients) [13]. It is still considered a state-of-the-art alorithm for generating textures. One can easily interpolate textures by interpolating the statistics used in the PS algorithm. However, from this statistical approach of texture, and due to the high number and complexity of statistics involved, it is hard to derive the distribution of a texture by just knowing all these numerical statistics. Yet the probabilistic modeling of visual perception lead to the fact that stimuli are best seen seen as distributions. In fact, by modeling textures as distribution, it would be theoretically possible to derive the perceptual scale from the Fisher Information of the generative model of the textures.

Another approach is to consider a particular texture directly as the realization of a distribution, aligning with the probabilistic hypothesis of textures as stationary random fields. Under certain assumptions (such as Gaussian distributions), well-established techniques exist for interpolating between different distributions. However, textures are high-dimensional objects with complex structures, and accurately defining their distributions presents significant challenges due to the sheer volume of data and the intricate relationships between texture features. Deep learning, particularly Variational Autoencoders (VAEs), offers a powerful solution to this problem. VAEs excel in learning a compact, lower-dimensional latent space where each point represents a distribution over texture features. By training on a large dataset of textures, VAEs learn to capture and encode complex, high-dimensional relationships into this latent space. Consequently, VAEs simplify the task of modeling and interpolating texture distributions by translating the high-dimensional problem into a lower-dimensional space where smooth and coherent transitions between textures are more easily achieved. In Section 3, we will thoroughly explore various VAE variants that will be employed for texture generation.

2 Testing Hypothesis under MLDS

2.1 Standard approach

Under MLDS the experimenter selects M stimuli, $s_1, \ldots, s_M$, each associated with a real number, v_i, such that $v_1 < v_2 < \ldots < v_M$. These values are the physical parameter of the stimuli. The experimenter presents the observer with quadruples, $(s_i, s_j; s_k, s_l)$, and asks which pair (s_i, s_j) or (s_k, s_l) is further apart. The experimenter's objective is to find

out the perceptual scale values $\phi(s_1), \ldots, \phi(s_M)$, corresponding to a discretization of the function ϕ, such that the observer judges (s_i, s_j) to be further apart than (s_k, s_l) when:

$$\Delta_{i,j,k,l} = \phi(s_i) - \phi(s_j) - \phi(s_k) + \phi(s_l) + N > 0 \tag{5}$$

where $N \sim \mathcal{N}(0, \sigma)$. The probability that the observer selects the first interval is:

$$P[\Delta_{i,j,k,l} > 0 \mid \phi(s_2), \ldots, \phi(s_{M-1}), \sigma] \tag{6}$$

where $\phi(2), \ldots, \phi(M-1)$ and σ are the parameters of the random variables $\Delta_{i,j,k,l}$. Note that since we are interested in the shape of ϕ and not in its absolute values, we can set $\phi(s_1) = 0$ and $\phi(s_M) = 1$. Since $\phi(s_i)$ are deterministic values, this is simply $\Phi_\sigma(\phi(s_i) - \phi(s_j) - \phi(s_k) + \phi(s_l))$ where

$$\Phi_\sigma(x) = P[-N \leq x] = P[N \leq x] = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{x} e^{-\frac{u^2}{2\sigma^2}}\, du \tag{7}$$

is the cumulative distribution function of the Gaussian random variable $N \sim \mathcal{N}(0, \sigma)$. If the observer instead selects the second interval, then the probability that he does so is just $1 - \Phi_\sigma(\phi(s_i) - \phi(s_j) - \phi(s_k) + \phi(s_l))$. For each trial t, we define $\Delta_t = \phi(s_{i_t}) - \phi(s_{j_t}) - \phi(s_{k_t}) + \phi(s_{l_t}) + N_t$. If we code the choice of the first interval as $R = 1$ and the choice of the second interval as $R = 0$ then the probability of the response pattern $(R_t)_{t=1,\ldots,T}$ across T trials is:

$$P(R_1, \ldots, R_T \mid \phi_2, \ldots, \phi_{M-1}, \sigma) = \prod_{t=1}^{T} \Phi_\sigma(\Delta_t)^{R_t} [1 - \Phi_\sigma(\Delta_t)]^{1-R_t}, \tag{8}$$

which is the likelihood function $L(\phi(s_2), \ldots, \phi(s_{M-1}), \sigma \mid R_1, \ldots, R_T)$. We estimate the parameters $\phi(s_2), \ldots, \phi(s_{M-1})$ and σ by maximizing the loglikelihood. The pseudo-algorithm 1 shows all important steps. In the Appendix A, you will find the main functions for an implementation in Python.

2.2 Psychometric function

The equation of perception $M(s) = \phi(s) + N$ can be understood like that: when an observer observes a stimulus s, the internal representation that he makes of s is noisy, i.e if s was slightly different he might not perceive it as different. A standard approach to calculate the sensitivity of an observer to a given change in s is a 2 alternative forced choice (2AFC) experiment - the observer is asked what is more "something" between two choices. An example could be to choose between two lines which one is more vertical. By repeating this experience several times with different values, one can derive the psychometric function, which is a measure of how sensitive the observer is to a certain parameterized stimulus. The psychometric function is defined as the probability that the internal representation of the test stimulus s_t is perceived as "greater" (for instance more vertical) than that of the reference stimulus s_i:

$$\psi_{s_i} : \begin{cases} S \subseteq \mathbb{R} & \longrightarrow \quad [0, 1] \\ s_t & \longmapsto \quad P(M(s_t) > M(s_i)) \end{cases}$$

Algorithm 1 MLDS Experiment

Initialize:

Select M stimuli, $s_1, s_2, \ldots, s_M$ with associated values $v_1, v_2, \ldots, v_M$

Set $\phi(s_1) = 0$ and $\phi(s_N) = 1$

Initialize $\phi(s_2), \phi(s_3), \ldots, \phi(s_{M-1})$ with initial guesses

Initialize σ with an initial guess

Set T as the number of trials

Generate quadruples:

for each trial t from 1 to T **do**

  Randomly select quadruples $(s_i, s_j; s_k, s_l)$

  Present quadruples to the observer

  Record observer's response R_t (1 if (s_i, s_j) is chosen, 0 otherwise)

end for

Compute likelihood:

Define the loglikelihood function:

$$L(\phi(s_2), \phi(s_3), \ldots, \phi(s_{M-1}), \sigma \mid R_1, R_2, \ldots, R_T) = \sum_{t=1}^{T} \left[\log(\Phi_\sigma(\Delta_t)^{R_t}) + \log((1 - \Phi_\sigma(\Delta_t))^{1-R_t})\right]$$

Maximize loglikelihood:

Use optimization method to find optimal $\phi(s_2), \phi(s_3), \ldots, \phi(s_{M-1})$, and σ by maximizing the likelihood function. [Note: In Python we can use the function "minimize" from "scipy" with constraints $0 = \phi(s_1) \leq \phi(s_2) \leq \cdots \leq \phi(s_{M-1}) \leq \phi(s_M) = 1$]

Output results:

Return the estimated perceptual scale values $\phi(s_1), \phi(s_2), \ldots, \phi(s_M)$

Return the estimated standard deviation σ

In practice, the probability $P(M(s_t) > M(s_i))$ is approximated empirically by: [number of times the observer choose $M(s_t)$ over $M(s_i)$]/ [number of trials].

Note that the experiment done in MLDS is a 2AFC, so it is possible to derive a psychometric function corresponding to the MLDS experience. In fact lets consider a slightly different version of the MLDS experiment than presented before where we present the observers with triples $(s_i, s_t; s_k)$ where s_i and s_j are fixed and will be our reference stimuli. By repeatedly asking which of the two pairs (s_i, s_t) or (s_t, s_j) is further apart, we can construct the following psychometric function:

$$\psi_{s_{i,j}}(s_t)\colon \begin{cases} S \subseteq \mathbb{R} & \longrightarrow \quad [0,1] \\ s_t & \longmapsto \quad P\left(M(s_i) - M(s_t) > M(s_t) - M(s_j)\right) \end{cases}$$

We notice that $\psi_{s_{i,j}}(s_t) = P(\Delta_{i,j,k,l} > 0)$, which means that this psychometric function assumes H1 (where we lift the absolute values). Alternatively, we could assume H2 (where we need to keep the absolute values) and define the psychometric function as $\tilde{\psi}_{s_{i,j}}(s_t) = P\left(|M(s_i) - M(s_t)| > |M(s_t) - M(s_j)|\right) = P(\widetilde{\Delta}_{i,j,k,l} > 0)$

2.3 Method for testing MLDS's assumption

To test whether H1 (lifting the absolute values by assuming the inherent noise of a stimulus can be ignored) or H2 (keeping the absolute values) is the better hypothesis, one can follow these steps:

1. Choose a well-known perceptual scale function ϕ.
2. Select your reference stimuli s_i and s_j
3. Construct the psychometric function $\psi_{s_{i,j}}$ by calculating, for several test stimuli s_t,

   $$\psi_{s_{i,j}}(s_t) = P\left(\phi(s_i) + N_i - (\phi(s_t) + N_t) > \phi(s_j) + N_j - (\phi(s_t) + N_t)\right),$$

   using Monte Carlo simulations.
4. Similarly, construct the psychometric function $\tilde{\psi}_{s_{i,j}}$
5. Construct the true psychometric function ψ_{true} by performing a 2AFC experiment
6. Compare the constructed psychometric functions $\psi_{s_{i,j}}$ and $\tilde{\psi}_{s_{i,j}}$ with the true one ψ_{true} and determine which one is closest.

If ψ_{true} is closer to $\psi_{s_{i,j}}$ (mathematically this could be calculated using the $L2$ norm of the difference) then H1 is the better assumption, otherwise H2 is.

We have constructed $\psi_{s_{i,j}}$ and $\tilde{\psi}_{s_{i,j}}$ for the following perceptual scale functions:

$$\text{s\_curve}(x) = \frac{1}{1 + e^{-10(x-0.5)}}, \quad \text{log\_curve}(x) = \log\left(1 + x(e-1)\right), \quad \text{quadratic\_curve}(x) = x^2$$

On the figure **??** ,we show the psychometric functions obtained under H1 and H2.

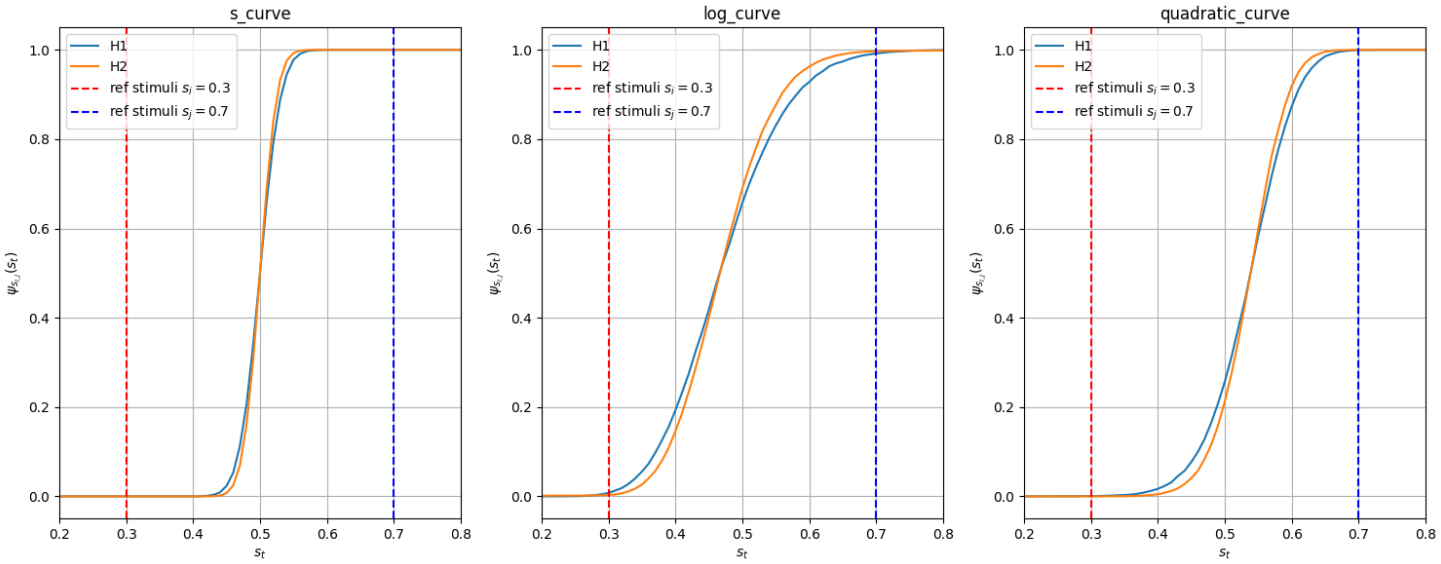

Figure 5: Psychometric functions: H1 vs H2

2.4 Importance of the right model

We can empirically test how well the MLDS method can infer perceptual scale functions assuming H1 versus H2. We used s_curve, log_curve, quadratic_curve as test functions to be inferred. For each of these test functions, we have generated two sets of simulated responses to an MLDS experience. To numerically generate the sets SR_1 (assumes H1) and SR_2 (assumes H2), we proceed as follows:
Generation of set SR_1:

1. For each trial t, sample the noise term $N_t \sim \mathcal{N}(0, \sigma^2)$.
2. Compute $\Delta_{i,j,k,l} = \phi(s_i) - \phi(s_j) - \phi(s_k) + \phi(s_l) + 2N_t$.
3. Set $R_t = 1$ if $\Delta_{i,j,k,l} > 0$; otherwise, set $R_t = 0$. The set SR_1 is composed of these responses across all trials.

Generation of set SR_2:

1. For each trial t, sample independent noise terms N_1 and N_2 from Gaussian distributions $N_1 \sim \mathcal{N}(0, \sigma^2)$ and $N_2 \sim \mathcal{N}(0, \sigma^2)$, respectively.
2. Compute $\widetilde{\Delta}_{i,j,k,l} = \left|\phi(s_i) - \phi(s_j) + \sqrt{2}N_1\right| - \left|\phi(s_k) - \phi(s_l) + \sqrt{2}N_2\right|$.
3. Set $R_t = 1$ if $\widetilde{\Delta}_{i,j,k,l} > 0$; otherwise, set $R_t = 0$. The set SR_2 is composed of these responses across all trials.

Note that when dealing with absolute differences between two Gaussian noise terms, the variance of the resulting difference is $2\sigma^2$, where σ^2 is the variance of the individual noise terms. Therefore, the standard deviation of the difference is $\sqrt{2}\sigma$.

As we can see in Figure 6 the standard MLDS performs better when inferring on SR_1. This is in line with our expectations (the standard MLDS assumes H1) and shows that

the underlying assumption H1 is indeed impactful. In other words, if H2 is true, then the standard MLDS method is not a good method to infer a perceptual scale function.

I attempted to derive a closed-form formula for the cumulative distribution function (CDF) of $|N_1| - |N_2|$, where N_1 and N_2 are independent Gaussian random variables. This formula would have been used as an alternative to Φ in Equation 8 for calculating the likelihood under assumption H2. Despite formal calculations performed using Python, no closed-form solution was found.

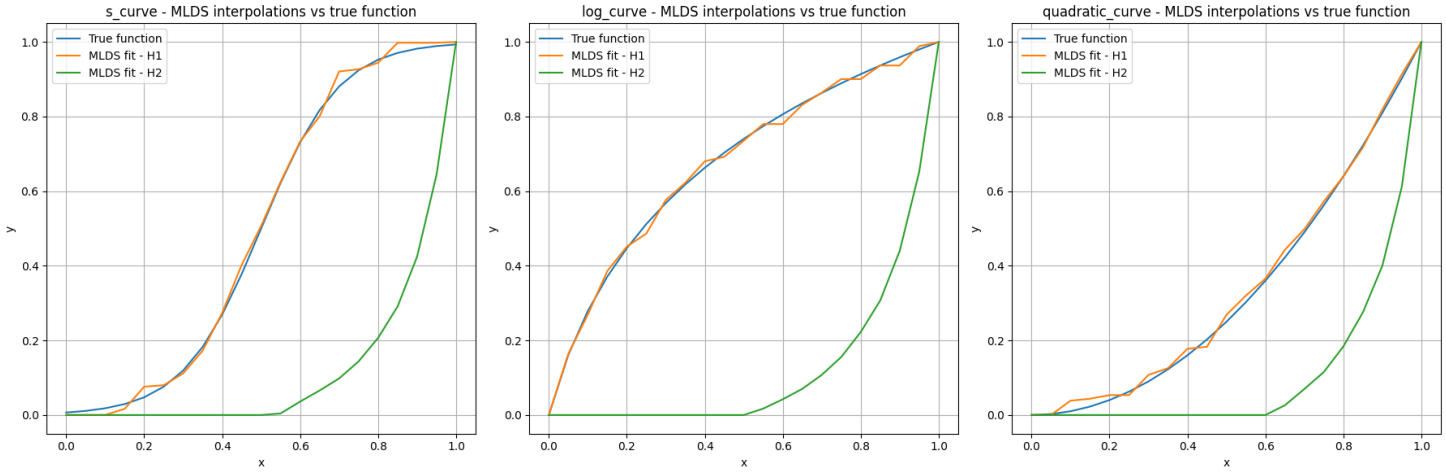

Figure 6: MLDS interpolation

3 Texture Synthesis with deep learning

In this section, we will discuss how VAE can help generate textures for MLDS experiments.

3.1 Theory of VAE

3.1.1 From autoencoder to VAE

A Variational Autoencoder (VAE) extends the traditional autoencoder by incorporating probabilistic elements. In a standard autoencoder, we have:

- An encoder that maps input data $\mathbf{x} \in \mathbb{X}$ to a latent representation $\mathbf{z} \in \mathbb{L}$. In our case, $\mathbb{X} = \mathbb{R}^{c \times h \times w}$ and $\mathbb{L} = \mathbb{R}^{D_L}$, where c represents the number of channels in the input (e.g., 3 for RGB images), h denotes the height, and w denotes the width of the input image.
- A decoder that reconstructs the input data from this latent representation.

Autoencoders are used to compress high-dimensional data into a lower-dimensional latent space. The latent variables are part of the model but are unobserved. For $\mathbf{x} \in \mathbb{X}$, the encoder computes $\mathbf{z} = f_\phi(\mathbf{x}) \in \mathbb{R}^{D_L}$, where $D_L \ll \dim(\mathbb{X})$ and f_ϕ is a neural network parameterized by ϕ. To ensure that $\mathbf{z}$ captures the essential information of $\mathbf{x}$, we train a generative neural network $g_\theta : \mathbb{R}^{D_L} \to \mathbb{X}$ to reconstruct $\mathbf{x}$ from $\mathbf{z}$. The cost function

ensures that $g_\theta(\mathbf{z}))$ is close to $\mathbf{x}$. Usually the following L2 norm is used to express the cost function:

$$L(\Theta, (\mathbf{x}_i)_{1\leq i\leq N}) = \sum_{i=1}^{N} \|\mathbf{x}_i - g_\theta(f_\phi(\mathbf{x}_i))\|_2^2,$$

where $\Theta = (\phi, \theta)$ and $(\mathbf{x}_i)_{1\leq i\leq N}$ is the observed data.

In a Variational Autoencoder (VAE), we introduce a probabilistic framework. The data $(\mathbf{x}_i)_{1\leq i\leq N}$ are assumed to be sampled from an unknown distribution $P(X)$, and the latent space is modeled as another unknown distribution $P(Z)$ over $\mathbb{R}^{D_L}$. Instead of mapping inputs to a single point in the latent space, the encoder maps inputs to a distribution from which latent variables $\mathbf{z}$ are sampled.

3.1.2 Evidence Lower Bound (ELBO)

To approximate the unknown joint distribution $P(X, Z)$, we use variational inference. This involves choosing a class of parameterized functions and finding the parameters that best approximate the true probability distribution, based on the available data. The parameterized function should be sufficiently flexible so that the approximated distribution is as close as possible to the true distribution. In our case, to train the VAE, we need to optimize the parameters Θ for the generative model $P_{X,Z|\Theta}(\mathbf{x}, \mathbf{z})$. The generative model defines the relationship between the observed data $\mathbf{x}$ and the latent variables $\mathbf{z}$, and is parameterized by the encoder f_ϕ that encode $\mathbf{x}$ into $\mathbf{z}$ and the decoder g_θ that reconstruct $\mathbf{x}$ from $\mathbf{z}$.

In practice, we aim to maximize the log-likelihood of the observed data, but direct optimization is intractable due to the integral over the unknown latent variables:

$$\log P_{X|\Theta}(\mathbf{x}) = \log \int P_{X,Z|\Theta}(\mathbf{x}, \mathbf{z})\, d\mathbf{z}. \tag{9}$$

The trick that we use to solve this issue, is to rewrite the log-likelihood as an expectation with respect to a paramtetrized approximate posterior distribution $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$, where Φ is the parameter of this approximate posterior distribution that we are trying to learn (variational inference). It comes:

$$\log P_{X|\Theta}(\mathbf{x}) = \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log P_{X|\Theta}(\mathbf{x})\right] \tag{2.5}$$

Equation (2.5) follows from the fact that the log-likelihood can be expressed as an expectation with respect to any random variable that is independent from the random variable X.

Next, we use the definition of conditional probability to expand the log-likelihood inside the expectation:

$$\log P_{X|\Theta}(\mathbf{x}) = \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{P_{X,Z|\Theta}(\mathbf{x},\mathbf{z})}{P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})}\right)\right] \tag{2.6}$$

$$= \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{P_{X,Z|\Theta}(\mathbf{x},\mathbf{z})}{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\cdot\frac{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}{P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})}\right)\right] \tag{2.7}$$

$$= \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{P_{X,Z|\Theta}(\mathbf{x},\mathbf{z})}{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\right)\right] + \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}{P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})}\right)\right] \tag{2.8}$$

$$= \mathcal{L}_{ELBO}(\Theta,\Phi;\mathbf{x}) + D_{KL}(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})\|P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})) \tag{2.9}$$

The first term $\mathcal{L}_{ELBO}(\Theta,\Phi;\mathbf{x})$ is known as the Evidence Lower Bound (ELBO), and the second term is the Kullback-Leibler (KL) divergence between the approximate posterior $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$ and the true posterior [2] $P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})$. The KL divergence, defined as

$$D_{KL}(\hat{P} \parallel P) = \int \hat{P}(\mathbf{z}) \log \frac{\hat{P}(\mathbf{z})}{P(\mathbf{z})}\, d\mathbf{z}, \tag{10}$$

is a measure of how one probability distribution diverges from a second, reference probability distribution.

Since the KL divergence is always non-negative, i.e., $D_{KL}(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})\|P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})) \geq 0$, it follows that:

$$\log P_{X|\Theta}(\mathbf{x}) \geq \mathcal{L}_{ELBO}(\Theta,\Phi;\mathbf{x}) \tag{2.10}$$

Thus, the ELBO $\mathcal{L}_{ELBO}(\Theta,\Phi;\mathbf{x})$ serves as a lower bound on the log-likelihood of the observed data. Maximizing the ELBO, by tuning parameter Φ and Θ, simultaneously achieves two things:

1. maximizes the likelihood of the observed data $\log P_{X|\Theta}(\mathbf{x})$,
2. minimizes the KL divergence between the approximate posterior and the true posterior.

Both objectives are achieved simultaneously because:

1. The KL divergence is non-negative, so it cannot compensate indefinitely for increases in the ELBO.
2. The optimization is performed over both parameters Θ and Φ. When Φ is optimized, the log-likelihood $\log P_{X|\Theta}(\mathbf{x})$, which does not depend on Φ, remains constant. Therefore, the KL divergence must decrease.

[2] The term "true" here refers to the posterior distribution we are approximating. However, it is important to note that "true" is somewhat misleading, as the true distribution of the data is never fully known or achieved. Instead, we are aiming to approximate it, and this approximation inherently has limitations.

We can furter decompose $\mathcal{L}_{ELBO}(\Theta, \Phi; \mathbf{x})$:

$$\mathcal{L}_{ELBO}(\Theta, \Phi; \mathbf{x}) = \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})} \left[\log \frac{P_{X|Z,\Theta}(\mathbf{x}|\mathbf{z}) P_{Z|\Theta}(\mathbf{z})}{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})} \right] \tag{11}$$

$$= \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})} \left[\log P_{X|Z,\Theta}(\mathbf{x}|\mathbf{z}) \right] - \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})} \left[\log \frac{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}{P_{Z|\Theta}(\mathbf{z})} \right] \tag{12}$$

The first term represents the expected log-likelihood, often called the reconstruction term, and the second term is the Kullback-Leibler (KL) divergence between the approximate posterior $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$ and the prior $P_{Z|\Theta}(\mathbf{z})$.

$$\mathcal{L}_{ELBO}(\Theta, \Phi; \mathbf{x}) = \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})} \left[\log P_{X|Z,\Theta}(\mathbf{x}|\mathbf{z}) \right] - \mathrm{KL} \left(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x}) \| P_{Z|\Theta}(\mathbf{z}) \right) \tag{13}$$

3.1.3 Standard Gaussian assumptions

In the context of VAEs, we often make the following Gaussian assumptions to simplify the terms in the ELBO:

1. The prior distribution over the latent variable $\mathbf{z}$ is a standard normal distribution: $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$.
2. The approximate posterior distribution $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$ is Gaussian with parameters $\mu_\Phi(\mathbf{x})$ and $\Sigma_\Phi(\mathbf{x})$, meaning: $\mathbf{z}|\mathbf{x} \sim \mathcal{N}(\mu_\Phi(\mathbf{x}), \Sigma_\Phi(\mathbf{x}))$.
3. The likelihood $P_{X|Z,\Theta}(\mathbf{x}|\mathbf{z})$ is Gaussian: $\mathbf{x}|\mathbf{z} \sim \mathcal{N}(g_\Theta(\mathbf{z}), \mathbf{I})$, where g_Θ is a neural network.

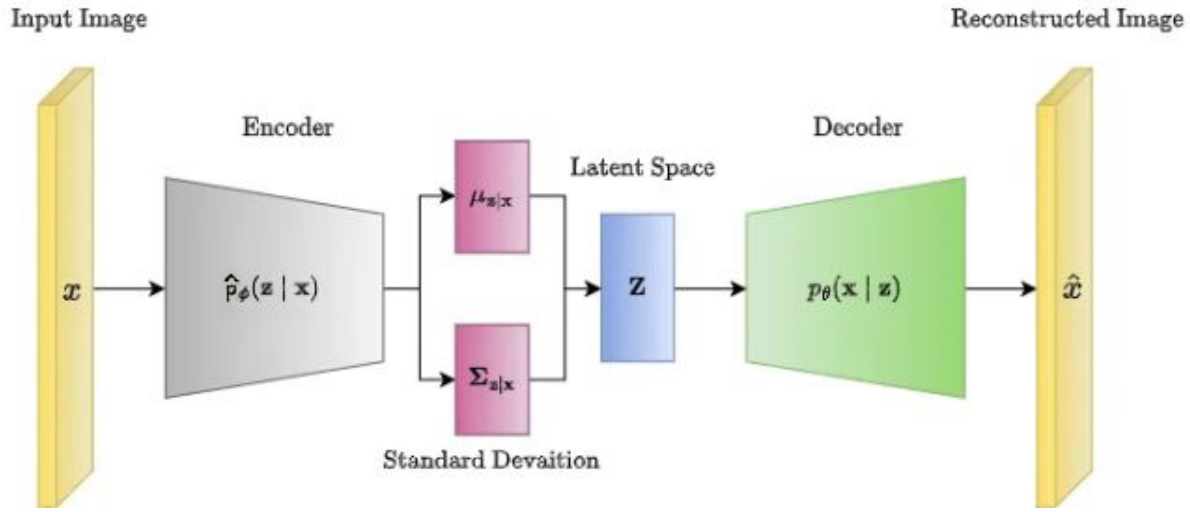

Figure 7: Standard VAE.

In the second assumption, instead of using traditional variational inference—where we infer the parameters of a Gaussian distribution separately for each data point $\mathbf{x}$—we use

a neural network to generate the parameters (the mean $\mu_\Phi(\mathbf{x})$ and the variance $\Sigma_\Phi(\mathbf{x})$) of a Gaussian distribution for all data points $\mathbf{x}$ at once. This technique, called amortized variational inference, reduces the computational cost by using a single global network parametrized by Φ to perform inference across the entire dataset, i.e to sample from $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$. This is also essential for performing gradient descent during optimization, since we can optimize Φ.

Without amortized variational inference, it would be impractical to perform separate inference for each data point, making gradient-based optimization methods difficult to apply. It also enhances efficiency by reducing the computational cost and memory usage associated with individual inference. Additionally, it allows the model to handle large datasets more effectively and improves generalization by enabling the global network to learn patterns across the entire dataset, leading to more accurate approximations of the posterior distribution.

Considering these assumptions, the log-likelihood term becomes:

$$\log P_{X|Z,\Theta}(\mathbf{x}|\mathbf{z}) = -\frac{1}{2}\left(\|\mathbf{x} - g_\Theta(\mathbf{z})\|_2^2 + D_L \log(2\pi)\right).$$

Taking the expectation with respect to $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$:

$$\mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log P_{X|Z,\Theta}(\mathbf{x}|\mathbf{z})\right] = -\frac{1}{2}\mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\|\mathbf{x} - g_\Theta(\mathbf{z})\|_2^2\right] - \frac{D_L}{2}\log(2\pi).$$

The KL divergence between two multivariate Gaussians [14] $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$ is given by:

$$\mathrm{KL}(\mathcal{N}(\mu_1, \Sigma_1)\|\mathcal{N}(\mu_2, \Sigma_2)) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_2^{-1}\Sigma_1) + (\mu_2 - \mu_1)^T\Sigma_2^{-1}(\mu_2 - \mu_1) - k + \log\frac{\det\Sigma_2}{\det\Sigma_1}\right).$$

For our case:

$$\mathrm{KL}(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})\|P_{Z|\Theta}(\mathbf{z})) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_\Phi(\mathbf{x})) + \mu_\Phi(\mathbf{x})^T\mu_\Phi(\mathbf{x}) - D_L - \log\det\Sigma_\Phi(\mathbf{x})\right).$$

Combining these, the final ELBO is:

$$\mathcal{L}_{ELBO}(\Theta, \Phi; \mathbf{x}) = -\frac{1}{2}\mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\|\mathbf{x} - g_\Theta(\mathbf{z})\|_2^2\right] - \frac{1}{2}\left(\mu_\Phi(\mathbf{x})^T\mu_\Phi(\mathbf{x}) + \mathrm{tr}(\Sigma_\Phi(\mathbf{x}))\right) - \frac{D_L}{2}\log(2\pi).$$

3.1.4 VAE algorithm

The training of a Variational Autoencoder (VAE) involves maximizing the Evidence Lower Bound (ELBO), which requires performing gradient descent with respect to both parameters Θ (decoder parameters) and Φ (encoder parameters). To achieve this, we need to compute gradients of the ELBO with respect to these parameters.

First, consider the gradient with respect to the decoder parameters Θ. The expectation in the ELBO is over the latent variable $\mathbf{z}$, which is sampled from the approximate posterior

distribution $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$. The gradient with respect to Θ is:

$$\nabla_\Theta \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[-\frac{1}{2}\|\mathbf{x} - g_\Theta(\mathbf{z})\|_2^2\right].$$

Since Θ only influences the generative model $g_\Theta(\mathbf{z})$ and not the distribution $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$, we can swap the gradient and expectation operators (Leibniz integral rule):

$$\mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[-\frac{1}{2}\nabla_\Theta\|\mathbf{x} - g_\Theta(\mathbf{z})\|_2^2\right].$$

This allows for efficient computation of the gradient with respect to Θ.

Now, consider the gradient with respect to the encoder parameters Φ. The ELBO's expectation is taken with respect to the distribution $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$, which depends on Φ. Therefore, the gradient is:

$$\nabla_\Phi \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[-\frac{1}{2}\|\mathbf{x} - g_\Theta(\mathbf{z})\|_2^2\right] - \frac{1}{2}\nabla_\Phi\left(\mu_\Phi(\mathbf{x})^T\mu_\Phi(\mathbf{x}) + \mathrm{tr}(\Sigma_\Phi(\mathbf{x}))\right).$$

However, in this case, the expectation and gradient operators cannot be swapped because the distribution $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$ depends on the parameter Φ. As a result, directly computing the gradient of the expectation is not straightforward. To address this issue, the reparameterization trick is employed. It involves expressing the latent variable $\mathbf{z}$ as:

$$\mathbf{z} = \mu_\Phi(\mathbf{x}) + \sigma_\Phi(\mathbf{x}) \cdot \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, I)$ is a standard normal noise term. This transformation separates the randomness (ϵ) from the parameters $\mu_\Phi(\mathbf{x})$ and $\sigma_\Phi(\mathbf{x})$. As a result, the dependency of $\mathbf{z}$ on Φ is removed from the expectation, allowing the gradient ∇_Φ to directly apply to the parameters $\mu_\Phi(\mathbf{x})$ and $\sigma_\Phi(\mathbf{x})$. This simplifies the computation, making the gradients with respect to Φ tractable and efficient to calculate.

In practice, instead of computing the full expectation, we approximate it by sampling from the distribution (also using the reparametrization trick). This Monte Carlo estimation is computationally efficient and provides an unbiased estimate of the gradient necessary for optimization.

The pseudo-algorithm 4 outlines the general steps for training a VAE:

3.1.5 Classic Architecture of a VAE

The classic architecture of a Variational Autoencoder (VAE) consists of an encoder and a decoder, each leveraging convolutional layers and fully connected layers. The encoder uses convolutional layers to reduce the spatial dimensions of the input image while increasing the depth of feature maps. It concludes with fully connected layers to output the parameters of the latent Gaussian distribution, specifically yielding $2 \cdot n$ dimensions where n is the latent space dimension (mean and log-variance). The decoder mirrors this structure, starting with fully connected layers that project the latent space representation to a higher-dimensional space. This is followed by transposed convolutional layers that upsample the representation back to the original input dimensions. Activation functions

Algorithm 2 Training a Variational Autoencoder (VAE)

Initialize:

Set initial values for the encoder parameters Φ and decoder parameters Θ

Set the number of training epochs n_epochs and the learning rate η

Initialize the variational distribution parameters (mean and covariance) for the approximate posterior

For each epoch:

for each batch of data $(x_1, x_2, \ldots, x_B)$ **do**

Forward Pass:

Compute the approximate posterior distribution $\mathbf{z}|\mathbf{x} \sim \mathcal{N}(\mu_\Phi(\mathbf{x}), \Sigma_\Phi(\mathbf{x}))$ for each data point x in the batch

Sample M latent variables $z^{(1)}, z^{(2)}, \ldots, z^{(M)}$ from the approximate posterior distribution

Reconstruction:

Compute the reconstruction $\hat{x}^{(m)} = g_\Theta(z^{(m)})$ for each sampled $z^{(m)}$

Compute ELBO:

Estimate the reconstruction loss using Monte Carlo approximation (empirical expectation):

$$\frac{1}{M}\sum_{m=1}^{M}\left(-\frac{1}{2}\|x - \hat{x}^{(m)}\|_2^2\right)$$

Compute the KL divergence between Gaussian distributions:

$$\mathrm{KL}\left(\hat{P}_{Z|X,\Phi}(z|x)\|P_{Z|\Theta}(z)\right) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_\phi(\mathbf{x})) + \mu_\phi(\mathbf{x})^T\mu_\phi(\mathbf{x}) - D_L - \log\det\Sigma_\phi(\mathbf{x})\right)$$

Compute the ELBO:

$$\mathcal{L}_{ELBO}(\Theta, \Phi; x) = \frac{1}{M}\sum_{m=1}^{M}\left(-\frac{1}{2}\|x - g_\theta(z^{(m)})\|_2^2\right) - \frac{1}{2}\left(\mu_\phi(\mathbf{x})^T\mu_\phi(\mathbf{x}) + \mathrm{tr}(\Sigma_\phi(\mathbf{x}))\right) - \frac{D_L}{2}\log(2\pi)$$

Backpropagation and Optimization:

Compute gradients of the ELBO with respect to Φ and Θ

Update the parameters Φ and Θ using the gradients and learning rate η

end for

Output results:

Return the trained encoder parameters Φ

Return the trained decoder parameters Θ

such as ReLU are applied after convolutional and transposed convolutional layers, with sigmoid or tanh functions often used in the output layer to match the data type.

The key parameters of convolutional and transposed convolutional layers include:

- **Kernel Size**: The dimensions of the convolutional or transposed convolutional filters (e.g., 3×3).
- **Stride**: Determines the step size of the kernel as it moves across the input, affecting the output dimensions (e.g., a stride of 2 means the kernel is moved by 2 pixels).
- **Padding**: Specifies whether to pad the input to preserve spatial dimensions or to reduce them.
- **Output Padding**: Used in transposed convolutions to adjust the size of the output feature map to match the desired dimensions.

To control the output dimension D_{out} for a convolutional layer, we use this formula:

$$D_{out} = \frac{D_{in} - F + 2P}{S} + 1 \tag{14}$$

where D_{in} is the input dimension, F is the filter size, P is the padding, and S is the stride.

For a transposed convolutional layer, the output dimension is:

$$D_{out} = S \cdot (D_{in} - 1) + F - 2P + \text{Output Padding} \tag{15}$$

where the parameters are similar, with output padding used to fine-tune the final size of the output.

3.1.6 Interpolation between textures

To generate intermediate textures for interpolation and perform MLDS experiments, it is crucial to choose an effective method. Pixel-wise interpolation involves blending the pixel values of two textures directly. Given two textures $\mathbf{x}_1$ and $\mathbf{x}_2$, the interpolated texture is obtained by linearly combining the pixel values from each texture. However, this method often leads to unrealistic results because it does not consider the underlying structure of the textures. The result can be visually incoherent and may produce artifacts or discontinuities, as it fails to account for the semantic relationships between texture features.

In contrast, the primary advantage of using a VAE for interpolating textures is that it leverages a structured latent space $\mathcal{L}$. The VAE maps textures to this latent space through an encoder network, where similar textures are located close to each other. The encoder produces a distribution $\hat{P}_{\mathcal{Z}|\mathbf{X},\Phi}(\mathbf{z}|\mathbf{x})$ for each texture, representing the likelihood that a texture corresponds to a point in $\mathcal{L}$.

The VAE training process minimizes the Kullback-Leibler (KL) divergence between this learned distribution and a prior distribution $P_{\mathcal{Z}}(\mathbf{z})$ over the latent space, $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$ in our case. This encourages the latent space to be continuous and well-structured, ensuring that similar textures are mapped to nearby points in $\mathcal{L}$, and making the space dense and compact. By interpolating between two points in the latent space $\mathcal{L}$, we can

generate intermediate textures that are consistent and smooth. This approach preserves the underlying structure of the textures and produces more realistic transitions compared to pixel-wise interpolation, as it respects the learned features and relationships in the latent space.

When dealing with Gaussian distributions, interpolation can be performed by directly manipulating the parameters of the distributions.

Multivariate Gaussian Distribution A multivariate Gaussian distribution in n dimensions is defined by a mean vector $\boldsymbol{\mu} \in \mathbb{R}^n$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$. The probability density function (PDF) of a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by:

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right)$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is a random vector.

The covariance matrix $\boldsymbol{\Sigma}$ must be positive semi-definite (PSD), meaning that for any vector $\boldsymbol{v} \in \mathbb{R}^n$, the quadratic form $\boldsymbol{v}^\top \boldsymbol{\Sigma} \boldsymbol{v} \geq 0$. This condition ensures that the variance along any direction $\boldsymbol{v}$ in the vector space is non-negative, which is essential because variance, as a measure of spread, cannot be negative. A PSD covariance matrix guarantees that the Gaussian distribution is valid, preventing non-physical scenarios such as negative variances.

Linear Interpolation Between Two Gaussians When interpolating between two multivariate Gaussian distributions $\mathcal{N}_1(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $\mathcal{N}_2(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, a simple method is to perform a linear interpolation. The interpolated mean vector $\boldsymbol{\mu}_\alpha$ and covariance matrix $\boldsymbol{\Sigma}_\alpha$ are defined as:

$$\boldsymbol{\mu}_\alpha = (1 - \alpha)\boldsymbol{\mu}_1 + \alpha \boldsymbol{\mu}_2$$
$$\boldsymbol{\Sigma}_\alpha = (1 - \alpha)\boldsymbol{\Sigma}_1 + \alpha \boldsymbol{\Sigma}_2$$

where $\alpha \in [0, 1]$. While this approach is straightforward, it may not guarantee that the interpolated covariance matrix $\boldsymbol{\Sigma}_\alpha$ remains positive semi-definite. This is because a linear combination of PSD matrices is not necessarily PSD, which could result in an invalid covariance matrix for the interpolated Gaussian.

Geodesic Interpolation To ensure that the interpolated covariance matrix remains positive definite, geodesic interpolation is used. In this approach, distributions are treated as points on a curved surface. The geodesic, which is the shortest path between two points on this curved surface, represents the most natural way to interpolate between covariance matrices. By following this path, we ensure that the interpolated covariance matrix remains valid and positive definite throughout the process.

For two covariance matrices $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$, the geodesic interpolation of the covariance matrix is given by this formula [11]:

$$\boldsymbol{\Sigma}_\alpha = \boldsymbol{\Sigma}_1^{1/2} \left(\boldsymbol{\Sigma}_1^{-1/2} \boldsymbol{\Sigma}_2 \boldsymbol{\Sigma}_1^{-1/2}\right)^\alpha \boldsymbol{\Sigma}_1^{1/2}$$

where $\boldsymbol{\Sigma}_1^{1/2}$ denotes the matrix square root of $\boldsymbol{\Sigma}_1$, and $\alpha \in [0, 1]$ is the interpolation

parameter.

For the mean vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ of the two distributions, linear interpolation is typically used:

$$\boldsymbol{\mu}_\alpha = (1-\alpha)\boldsymbol{\mu}_1 + \alpha\boldsymbol{\mu}_2$$

Here, $\alpha \in [0,1]$ smoothly transitions between the two means, ensuring that the interpolated mean vector is a valid combination of the two original means.

Combining these interpolations, the interpolated Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_\alpha, \boldsymbol{\Sigma}_\alpha)$ has a mean vector $\boldsymbol{\mu}_\alpha$ and a covariance matrix $\boldsymbol{\Sigma}_\alpha$ that are both valid, with the covariance matrix remaining positive definite throughout the interpolation process.

3.2 Addressing blurriness of images generated by standard VAE

3.2.1 Explanation

VAEs often generate blurry images, even when the training data consists of sharp images. This is also the case with the textures that we have generated. Since our goal is to create textures that closely resemble natural ones, this blurriness poses a significant problem. If the generated textures lack sharpness, they may not be perceived accurately by the human eye, potentially leading to misleading results in MLDS experiments on texture perception.

This blurriness may be attributed to the optimization process, where the model maximizes the evidence lower bound (ELBO) and therefore minimizes the KL divergence between the true posterior and the approximate posterior:

$$\log P_{X|\Theta}(\mathbf{x}) = \mathcal{L}_{ELBO}(\Theta, \Phi; \mathbf{x}) + D_{KL}\left(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x}) \parallel P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})\right). \tag{16}$$

The KL divergence is not symmetric, meaning that $D_{KL}(\hat{P} \parallel P) \neq D_{KL}(P \parallel \hat{P})$. In the context of VAEs, this asymmetry plays a significant role during optimization. Specifically, minimizing $D_{KL}\left(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x}) \parallel P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})\right)$ tends to make the approximate posterior $\hat{P}_{Z|X,\Phi}$ converge to a distribution with lower variance compared to the "true" posterior. This effect is illustrated in Figure 8, where (a) shows $\min_\gamma D_{KL}(\hat{P}_\gamma \parallel P)$ over the parameterized distribution $\hat{P}_\gamma$, and (b) shows $\min_\gamma D_{KL}(P \parallel \hat{P}_\gamma)$. In the figure, the red circles represent the parameterized Gaussian distribution $\hat{P}_\gamma$ with an isotropic covariance matrix $(\gamma \cdot I)$, while the green ellipse represents the target distribution, a Gaussian distribution with a full covariance matrix.

As described by Bredell et al. [3], this incentivizes the model $P_{Z|X,\Theta}(\mathbf{z})$ to cover more of the latent space to avoid high penalties, leading to the learned joint distribution $P_{X,Z|\Theta}(\mathbf{x})$ having a large variance too. Consequently, the generated images, sampled from this broader distribution, are more diverse but often less sharp, leading to the characteristic blurriness observed in VAEs. This blurriness arises because the model, in trying to avoid penalization from the KL divergence, ends up producing a distribution that is too spread out, including regions that correspond to blurry reconstructions of the input data.

In the following sections, we will explore several methods to address this blurriness issue.

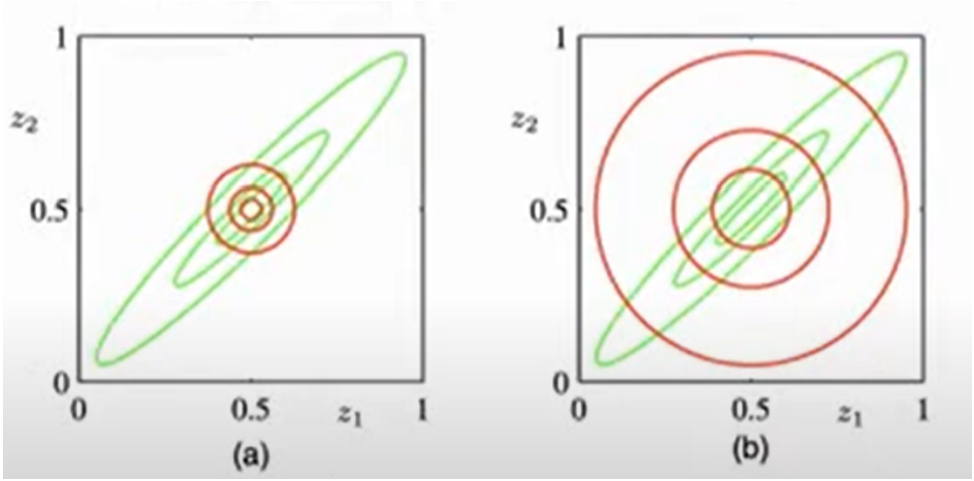

Figure 8: KL divergence optimun - source: Rianne van den Berg

3.2.2 Solution 1: VAE with Normalizing Flow

The first solution that we will explore and apply to the generation of textures is Normalizing Flow. Normalizing Flow is a method used to construct a complex probability distribution from a simple one (a uniform or Gaussian distribution) by applying a sequence of invertible and differentiable mappings. These conditions on the mapping should allow that the overall model remains tractable.

In the context of Variational Autoencoders (VAEs), normalizing flows can be used to enhance the expressiveness of the approximate posterior distribution $\hat{P}_{Z|X,\Phi}$. This method has been introduced by Rezende & Mohamed [5] By incorporating normalizing flows into the VAE framework, we can decrease the KL divergence between the approximate and the true posterior and hence achieve a tighter Evidence Lower Bound (ELBO):

$$\log P_{X|\Theta}(\mathbf{x}) = \mathcal{L}_{ELBO}(\Theta, \Phi; \mathbf{x}) + D_{KL}(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x}) \| P_{Z|X,\Theta}(\mathbf{z}|\mathbf{x})) \tag{17}$$

But more importantly for our blurriness problem, by increasing the expressiveness of the approximate posterior distribution, we avoid that minimizing the KL divergence leads to having a too high variance for $P_{Z|X,\Theta}(\mathbf{z})$.

Normalizing Flow - General Case Let $\mathbf{z}$ be a random variable defined in $\mathbb{R}^d$ with an associated probability distribution $p(\mathbf{z})$. To generate a more complex random variable, we apply a transformation $f : \mathbb{R}^d \to \mathbb{R}^d$, where f is a diffeomorphism (i.e., a function that is both invertible and continuously differentiable, with a continuously differentiable inverse). The resulting transformed random variable $\mathbf{z}' = f(\mathbf{z})$ will have a new distribution $p(\mathbf{z}')$ that can be derived using the change of variables formula for probability densities.

Specifically, the distribution of $\mathbf{z}'$ is given by:

$$p(\mathbf{z}') = p(\mathbf{z}) \left| \det \left(\frac{\partial f^{-1}}{\partial \mathbf{z}'} \right) \right|,$$

where $\frac{\partial f^{-1}}{\partial \mathbf{z}'}$ is the Jacobian matrix of the inverse transformation f^{-1} evaluated at $\mathbf{z}'$. Since f is a diffeomorphism, the Jacobian matrix of f^{-1} is the inverse of the Jacobian matrix of

f. Moreover $\det(A^{-1}) = \det(A)^{-1}$, so we have:

$$p(\mathbf{z}') = p(\mathbf{z}) \left| \det \left(\frac{\partial f}{\partial \mathbf{z}} \right) \right|^{-1},$$

where $\frac{\partial f}{\partial \mathbf{z}}$ is the Jacobian matrix of f evaluated at $\mathbf{z}$.

We can extend this idea by successively applying a series of transformations f_k to $\mathbf{z}$, resulting in a sequence of transformed variables $\mathbf{z}_k = f_k(\mathbf{z}_{k-1})$ for $k = 1, \ldots, K$, where $\mathbf{z}_0 = \mathbf{z}$. To derive the probability density $p_K(\mathbf{z}_K)$, we start by considering the sequence of transformations. For each transformation f_k, the change of variables formula tells us that:

$$p(\mathbf{z}_k) = p(\mathbf{z}_{k-1}) \cdot \frac{1}{\left| \det \left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right|}.$$

Iterating this process from $k = 1$ to $k = K$, we get:

$$p(\mathbf{z}_K) = p(\mathbf{z}_0) \prod_{k=1}^{K} \frac{1}{\left| \det \left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right|}.$$

Finally, taking the logarithm of $p(\mathbf{z}_K)$, we obtain:

$$\log p(\mathbf{z}_K) = \log p(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| \det \left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right|.$$

This expression shows how the density evolves as we apply each transformation in the sequence, with the determinant of the Jacobian playing a critical role in adjusting the density to account for the change of variables at each step.

In the context of normalizing flows the KL divergence term of the ELBO becomes:

$$\begin{aligned}
\mathrm{KL}\big(\hat{P}_{Z|X,\Phi}(\mathbf{z}_K|\mathbf{x}) \| P_{Z|\Theta}(\mathbf{z}_K)\big) &= \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_K|\mathbf{x})} \left[\log \hat{P}_{Z|X,\Phi}(\mathbf{z}_K|\mathbf{x}) - \log P_{Z|\Theta}(\mathbf{z}_K) \right] \\
&= \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_K|\mathbf{x})} \left[\log \hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x}) - \sum_{k=1}^{K} \log \left| \det \left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right| - \log P_{Z|\Theta}(\mathbf{z}_K) \right] \\
&= \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x})} \left[\log \hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x}) - \sum_{k=1}^{K} \log \left| \det \left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right| - \log P_{Z|\Theta}(\mathbf{z}_K) \right] \\
&= \mathrm{KL}\left(\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x}) \| P_{Z|\Theta}(\mathbf{z}_K) \right) - \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x})} \left[\sum_{k=1}^{K} \log \left| \det \left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right| \right].
\end{aligned}$$

Now, substituting back into the ELBO:

$$\begin{aligned}
\mathcal{L}_{\mathrm{ELBO}}(\Theta, \Phi; \mathbf{x}) = \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x})} \left[\log P_{X|Z,\Theta}(\mathbf{x}|\mathbf{z}_K) \right] - \mathrm{KL}\left(\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x}) \| P_{Z|\Theta}(\mathbf{z}_K) \right) \\
+ \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x})} \left[\sum_{k=1}^{K} \log \left| \det \left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right| \right].
\end{aligned}$$

Under the Gaussian assumptions, the ELBO simplifies to:

$$\mathcal{L}_{\text{ELBO}}(\Theta, \Phi; \mathbf{x}) = -\frac{1}{2}\mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x})}\left[\|\mathbf{x} - g_\Theta(\mathbf{z}_K)\|_2^2\right] - \frac{D}{2}\log(2\pi) - \text{KL}\left(\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x}) \| P_{Z|\Theta}(\mathbf{z}_K)\right) + \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x})}\left[\sum_{k=1}^{K} \log\left|\det\left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right)\right|\right].$$

Planar Flows In the context of normalizing flows, the ELBO involves computing the determinant of the Jacobian matrix of a transformation. To ensure that this computation is tractable and efficient, we use transformations where the Jacobian determinant is straightforward to calculate. One such transformation is the planar flow.

Given a latent vector $\mathbf{z}$, the planar flow transformation function $f(\mathbf{z})$ is defined as:

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u} \cdot \tanh(\mathbf{w}^T\mathbf{z} + b), \tag{18}$$

where $\mathbf{u}$, $\mathbf{w}$, and b are parameters learned during training.

To compute the Jacobian matrix J of the transformation $f(\mathbf{z})$ with respect to $\mathbf{z}$, we have:

$$J = \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} = I + \mathbf{u} \cdot \mathbf{w}^T \cdot \psi, \tag{19}$$

where ψ is defined as:

$$\psi = 1 - \tanh^2(\mathbf{w}^T\mathbf{z} + b). \tag{20}$$

To find the log-determinant of the Jacobian, we use the matrix determinant lemma [4]. For a matrix of the form $I + \mathbf{u}\mathbf{v}^T$, the determinant is given by:

$$\det(I + \mathbf{u}\mathbf{v}^T) = 1 + \mathbf{v}^T\mathbf{u}. \tag{21}$$

Applying this lemma to our case, where $\mathbf{v} = \psi \cdot \mathbf{w}$, we get:

$$\text{log-det}(J) = \log\left|1 + \mathbf{w}^T\psi \cdot \mathbf{u}\right|. \tag{22}$$

Training a VAE with Planar Flows The pseudo-algorithm 3 outlines the general steps for training a VAE with planar flows:

3.2.3 Solution 2: VAE combined with GAN

The second solution we will explore involves leveraging the benefits of Generative Adversarial Networks (GANs).

Introduction to VAE GANs are a class of machine learning frameworks designed for generative modeling (as VAEs). They consist of two neural networks, the generator G and the discriminator D, which are trained adversarially.

- **Generator (G)**: The generator aims to produce data samples that are indistinguishable from real data. It maps a noise vector $\mathbf{z}$ (typically sampled from a known

Algorithm 3 Training a VAE with Planar Flows

Initialize:

Set initial values for encoder parameters Φ and decoder parameters Θ

Set number of epochs n_{epochs} and learning rate η

Initialize the number of flow steps K

Initialize planar flow parameters $\mathbf{u}_k$, $\mathbf{w}_k$, and b_k for each flow step k

For each epoch:

for each batch of data $(x_1, x_2, \ldots, x_B)$ **do**

Forward Pass:

Compute the transformed approximate posterior $\mathbf{z}|\mathbf{x} \sim f_K(\mathcal{N}(\mu_\Phi(\mathbf{x}), \Sigma_\Phi(\mathbf{x})))$

Sample M latent variables $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(M)}$ from the transformed approximate posterior

Reconstruction:

Compute the reconstruction $\hat{\mathbf{x}}^{(m)} = g_\Theta(\mathbf{z}^{(m)})$ for each sampled $\mathbf{z}^{(m)}$

Compute ELBO:

$\mathcal{L}_{\text{ELBO}}(\Theta, \Phi, \mathbf{u}, \mathbf{w}, \mathbf{b}; \mathbf{x}) = \text{Reconstruction Error} - \text{KL Divergence} + \text{Log-Determinant Term}$

where,

$$\text{Log-Determinant Term} = \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}_0|\mathbf{x})}\left[\sum_{k=1}^{K} \log\left|1 + \mathbf{u}_k^T \psi_k\right|\right].$$

where psi_k is $1 - \tanh^2(\mathbf{w}_k^T \mathbf{z}^{(k-1)} + b_k)$

Backpropagation and Optimization:

Compute gradients of the ELBO with respect to Φ and Θ

Update parameters Φ and Θ using gradients and learning rate η

end for

Output results:

Return trained encoder parameters Φ

Return trained decoder parameters Θ

Gaussian distribution) to the data space (in our case, the texture space). Formally, the generator function is denoted as:

$$\mathbf{x}_{\text{gen}} = G(\mathbf{z}; \Theta_G) \tag{23}$$

where Θ_G represents the parameters of the neural network acting as generator.

- **Discriminator (D)**: The discriminator evaluates whether a given sample is from the real data distribution or generated by G. It outputs a probability score that indicates the likelihood of the input being real. The discriminator function is defined as:

$$D(\mathbf{x}; \Theta_D) \tag{24}$$

where Θ_D represents the parameters of the neural network acting as discriminator.

The training of GANs involves a two-player minimax game between the generator and the discriminator. The objective functions for G and D are defined as follows:

- Discriminator Objective:

$$\mathcal{L}_D = \mathbb{E}_{P_X(\mathbf{x})} [\log D(\mathbf{x}; \Theta_D)] - \mathbb{E}_{P_Z(\mathbf{z})} [\log(1 - D(G(\mathbf{z}; \Theta_G); \Theta_D))] \tag{25}$$

The discriminator aims to maximize this loss by tuning Θ_D, distinguishing between real and fake samples.

- Generator Objective:

$$\mathcal{L}_G = -\mathbb{E}_{P_Z(\mathbf{z})} [\log D(G(\mathbf{z}; \Theta_G))] \tag{26}$$

The generator aims to minimize this loss by tuning Θ_G, making the discriminator believe that generated samples are real.

The optimization process alternates between updating D and G to reach an equilibrium where D cannot distinguish between real and generated samples, and G produces high-quality samples that are indistinguishable from the real data.

$$\min_{\Theta_G} \max_{\Theta_D} \mathcal{L}_D + \mathcal{L}_G$$

VAE-GAN Hybrid Architecture To address the blurriness issue inherent in VAEs, a hybrid approach combining Variational Autoencoders (VAEs) with Generative Adversarial Networks (GANs) can be adapted as introduced by [8]. In fact, the discriminor D from the GAN can easily distinguish between real sharp textures and blur fake textures. We hope that the information gained by the discriminor D could be backpropagated towards the generator of the VAE.

In the VAE-GAN hybrid model, the VAE component captures the latent space representation and generates samples, while the GAN component refines these samples to achieve higher fidelity. The architecture typically consists of:

- Encoder (VAE): The encoder maps the input data $\mathbf{x}$ to a latent representation $\mathbf{z}$ by approximating the posterior distribution $\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$.

- Decoder/Generator (VAE/GAN): The decoder generates reconstructions $\hat{\mathbf{x}} = g_\Theta(\mathbf{z})$ from the latent representation $\mathbf{z}$. The objective is to maximize the ELBO, balancing between reconstruction accuracy and the KL divergence between the approximate posterior and the prior.

- Discriminator (GAN): The discriminator evaluates whether images are real (from the dataset) or fake (generated by the generator). It outputs a probability indicating the likelihood that the input image is real.

The training process of the VAE-GAN hybrid involves the following objectives:

- VAE Objective: The encoder and decoder/generator are trained by maximizing the Evidence Lower Bound (ELBO):

$$\mathcal{L}_{VAE}(\Theta_G, \Phi; \mathbf{x}) = \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log P_{X|Z,\Theta_G}(\mathbf{x}|\mathbf{z})\right] - \mathrm{KL}\left(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x}) \| P_Z(\mathbf{z})\right).$$

- GAN Objective 1: The decoder/generator is trained by minimizing the generator loss:

$$\mathcal{L}_G(\Theta_G; \mathbf{x}) = -\mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log D(G(\mathbf{z}; \Theta_G))\right].$$

- GAN Objective 2: The discriminator is trained by minimizing the discriminator loss:

$$\mathcal{L}_D(\Theta_D; \mathbf{x}) = -\mathbb{E}_{P_{X|Z,\Theta_G}(\mathbf{x}|\mathbf{z})}\left[\log D(\mathbf{x}; \Theta_D)\right] + \mathbb{E}_{\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})}\left[\log(1 - D(G(\mathbf{z}; \Theta_G)))\right]$$

These three objectives cannot be optimized simultaneously due to their competing goals. The VAE objective and GAN Objective 1 are both focused on generating high-quality data, and thus should be trained alternately to balance reconstruction accuracy with adversarial refinement. On the other hand, GAN Objective 2 serves a distinct purpose by enhancing the discriminator's efficiency in distinguishing real from generated images. In practice, to ensure the discriminator is robust, it is often trained more frequently than the generator, which helps in stabilizing the overall training process of the VAE-GAN hybrid model. PLease see the pseudo-algorithm 4.

Spectral Regularization in GANs The training of Generative Adversarial Networks (GANs) often involves instability. One reason is the way the generator is optimized. It can be shown [1] that the generator's objective can be reformulate as follows:

$$\min_{\Theta_G}\left(\frac{1}{2}\mathrm{KL}\left(P_X \middle\| \frac{P_X + P_G}{2}\right) + \frac{1}{2}\mathrm{KL}\left(P_G \middle\| \frac{P_X + P_G}{2}\right)\right) = \min_{\Theta_G} D_{JS}\left(P_X \| P_G\right)$$

where P_G is the distribution of the data generated by the generator G, which is parameterized by Θ_G and D_{JS} denotes the Jensen-Shannon (JS) divergence. However, the JS divergence is not a good measure, as it is equal to $\log(2)$ [1] when the supports of P_X and P_G are different. This causes the gradients to vanish and makes it hard for the generator to learn.

Algorithm 4 Training a VAE-GAN Hybrid Model

Initialize: Encoder parameters Φ, decoder/generator parameters Θ_G, discriminator parameters Θ_D.
for each epoch **do**
  for each batch of data $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_B)$ **do**
  VAE Step:
    Encode inputs: $\mathbf{z} \sim \hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x})$.
    Reconstruct inputs: $\hat{\mathbf{x}} = g_\Theta(\mathbf{z})$.
    Maximize ELBO: $\mathcal{L}_{VAE}(\Theta_G, \Phi)$.
  GAN Step 1:
    Update generator G: Minimize $\mathcal{L}_G(\Theta_G)$.
  GAN Step 2:
    Update discriminator D: Minimize $\mathcal{L}_D(\Theta_D)$.
  end for
end for
Return: Trained parameters Φ, Θ_G, Θ_D.

To mitigate this, Arjovsky et al. (2017) [2] introduced the Wasserstein distance as an alternative. The Wasserstein distance between two distributions P_X and P_G is given by:

$$W(P_X, P_G) = \inf_{\gamma \in \Pi(P_X, P_G)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

The Wasserstein distance provides a more meaningful gradient even when the distributions P_X and P_G have disjoint supports, leading to more stable and reliable training of GANs compared to using KL or JS divergence.

Using the Kantorovich-Rubinstein duality, this can be expressed as:

$$W(P_X, P_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{P_X(\mathbf{x})}[f(\mathbf{x})] - \mathbb{E}_{P_Z(\mathbf{z})}[f(G(\mathbf{z}; \Theta_G))]$$

The generator's objective then becomes:

$$\min_G \max_{D \in \mathcal{D}_K} \mathbb{E}_{P_X(\mathbf{x})}[D(\mathbf{x})] - \mathbb{E}_{P_Z(\mathbf{z})}[D(G(\mathbf{z}; \Theta_G))], \tag{27}$$

where $\mathcal{D}_K$ denotes the set of K-Lipschitz continuous functions. To ensure D remains within this class, we can use spectral normalization, a technique introduced by Miyato and al. [10].

Spectral normalization (SN) is a technique to enforce Lipschitz continuity in the discriminator. The spectral norm of a weight matrix W, denoted as $\sigma(W)$, is its largest singular value:

$$\sigma(W) = \sup_{\|h\|_2 = 1} \|Wh\|_2$$

To enforce the Lipschitz condition $K = 1$ in practice, we rescale the weight matrices by their spectral norm:

$$\tilde{W} = \frac{W}{\sigma(W)}$$

It follows:

$$\|\tilde{W}h\|_2 = \left\| \frac{Wh}{\sigma(W)} \right\|_2 = \frac{\|Wh\|_2}{\sigma(W)} \leq \frac{\sigma(W)\|h\|_2}{\sigma(W)} = \|h\|_2$$

Thus, $\|\tilde{W}h\|_2 \leq \|h\|_2$, implying that the Lipschitz constant for this layer is at most 1. Since spectral normalization is applied to each layer and the usual activation functions are 1-Lipschitz, the overall Lipschitz constant of the discriminator D, being the product of the Lipschitz constants of its layers, remains at most 1:

$$\|D(\mathbf{x}) - D(\mathbf{x}')\| \leq \prod_{l=1}^{L} \sigma(W_l)\|\mathbf{x} - \mathbf{x}'\| \leq \|\mathbf{x} - \mathbf{x}'\|$$

The training process for GAN with spectral normalization can be summarized in the pseudo-algorithm 5.

Algorithm 5 Training GAN with Spectral Normalization

1: **Initialize:** Generator parameters Θ_G, discriminator parameters Θ_D.
2: **for** each epoch **do**
3: **for** each batch of data $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_B)$ **do**
Discriminator Step:
4: Update D: Maximize $\mathcal{L}_D$ by tuning Θ_D after applying spectral normalization.
Generator Step:
5: Update G: Minimize $\mathcal{L}_G$ by tuning Θ_G.
6: **end for**
7: **end for**
8: **Return:** Trained parameters Θ_G, Θ_D.

3.2.4 Solution 3: Architectural Considerations

Here, we will investigate an innovative approach to mitigate the blurriness issue, with partial inspiration drawn from Rémi Gribonval's colloquium at MAP5 titled 'Frugality in Machine Learning: Sparsity as a Value for the Future.'

Large Sparse Kernels As we have seen in 3.1.5, the neural network is to a large extend made out of convolutional layers. A convolution operation in the spatial domain corresponds to a multiplication in the frequency domain. Let k be a convolution kernel and x be an input signal or image. The convolution operation $y(t)$ is given by:

$$y(t) = (k * x)(t)$$

where $*$ denotes the convolution operator.

In the frequency domain, the convolution theorem states that:

$$\mathcal{F}\{y(t)\} = \mathcal{F}\{k * x\} = \mathcal{F}\{k\} \cdot \mathcal{F}\{x\}$$

where $\mathcal{F}\{\cdot\}$ denotes the Fourier transform. Therefore, the frequency domain representation of a convolution is simply the pointwise multiplication of the Fourier transforms of

the kernel and the input signal.

For a discrete kernel $k[n]$ of size L, with $L \leq N$ where N is the length of the input signal, the Discrete Fourier Transform (DFT) is given by:

$$K(k) = \sum_{n=0}^{L-1} k[n] e^{-j\frac{2\pi}{N}kn}$$

where $K(k)$ represents the frequency response of the kernel at the frequency index k.

The magnitude of the DFT, $|K(k)|$, acts as a frequency filter, which can be either a low-pass filter, high-pass filter, or something in between, depending on the characteristics of the kernel $k[n]$.

- **Low-Pass Filter:** A kernel with smooth coefficients, such as $k[n] = \{1, 1, 1\}$, will have a DFT that primarily retains low-frequency components. The magnitude of its DFT is given by:
$$|K(\omega)| = \left|1 + e^{-i\omega} + e^{-i2\omega}\right|$$
This kernel smooths out variations, filtering out high frequencies and blurring details in the image.

- **High-Pass Filter:** A kernel designed to detect edges, such as $k[n] = \{-1, 1\}$, will have a DFT with higher gains at higher frequencies. The magnitude of its DFT is given by:
$$|K(\omega)| = \left|-1 + e^{-i\omega}\right| = 2\left|\sin\left(\frac{\omega}{2}\right)\right|$$
This kernel highlights rapid changes and edges in the image, enhancing details and reducing blurriness.

To effectively capture and analyze variations in frequency filtering, we need to examine how different kernels impact the frequency domain. To capture more information and observe variations in the types of frequencies filtered, it is crucial to analyze the frequency response of the kernel. A kernel that shows significant variation in its Fourier Transform (DFT) can filter out a wider range of frequencies. This is important for applications where distinguishing between different frequency components is necessary, as high-frequency components correlate with finer details and less blur.

To achieve a diverse frequency response, we should target kernels whose DFT exhibits substantial variation. One way to quantify this variability is through the Total Variation (TV) of the magnitude of the DFT. Total Variation is a measure of how much the magnitude of the DFT changes as a function of frequency. It is calculated as follows:

$$\text{Total Variation (TV)} = \sum_{i=1}^{M-1} ||K(\omega_{i+1})| - |K(\omega_i)||$$

where $|K(\omega_i)|$ represents the magnitude of the DFT at the i-th frequency, ω_i, and M is the total number of discrete frequency points.

Sparsity refers to the property of a matrix or tensor in which most of the elements are zero. In the context of convolutional kernels, a sparse kernel has a significant number of zero coefficients. Mathematically, if a kernel k has T total elements and t_s non-zero elements, its sparsity can be quantified as:

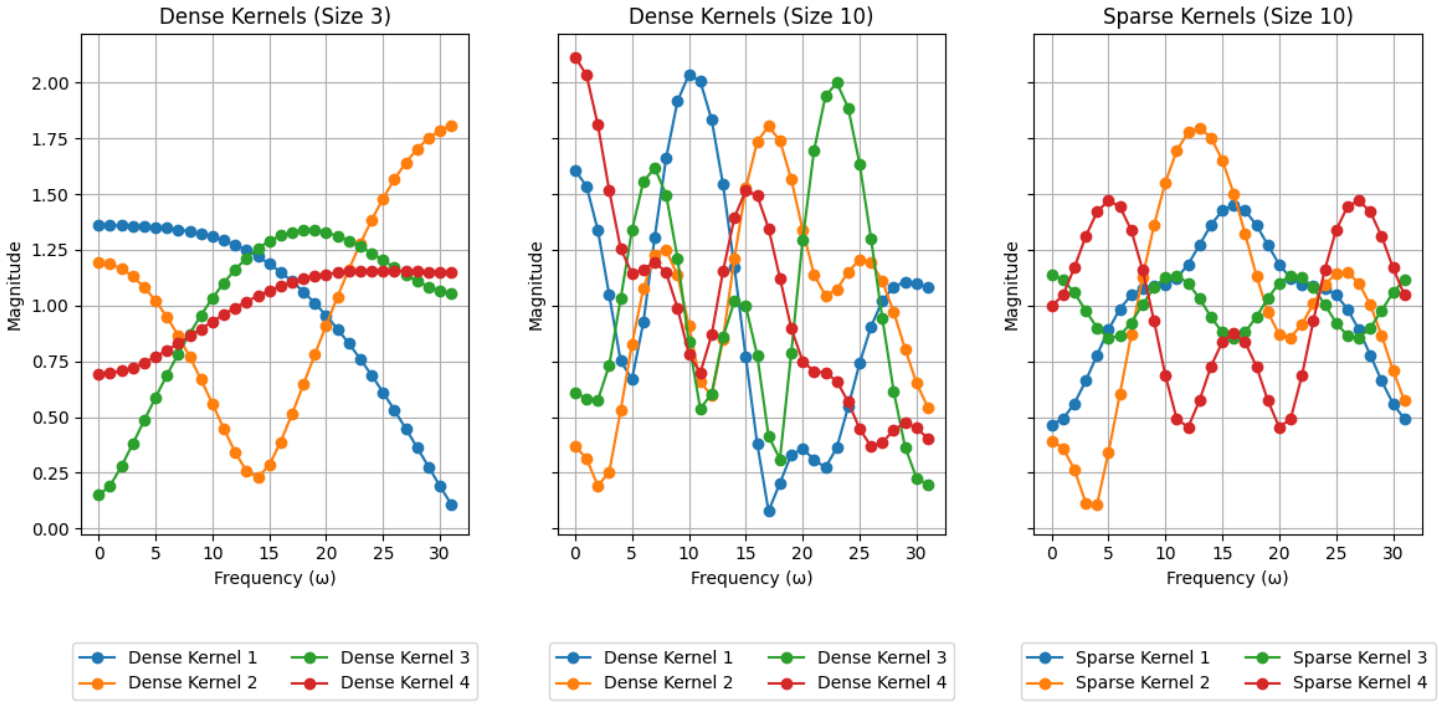

Figure 9: Comparison of frequency responses for different kernels

$$\text{Sparsity} = \frac{T - t_s}{T}$$

where Sparsity ranges from 0 (dense kernel) to 1 (fully sparse kernel).
Sparse kernels offer notable computational advantages:

1. Convolution operations with sparse kernels involve fewer non-zero multiplications compared to dense kernels, leading to reduced computational cost and faster execution times, especially for large-scale networks.

2. Sparse kernels require less memory storage since many of their elements are zero.

However, it's important to note that convolutions are often computed in the frequency domain for dense kernels, leveraging Fast Fourier Transform (FFT) to speed up the convolution process. This approach might not always directly benefit from sparsity as effectively, since FFT-based implementations are optimized for dense kernels. In libraries like PyTorch, while FFT-based convolution can be used for dense kernels, operations with sparse kernels are typically handled differently, and the actual computational advantage can depend on the implementation and the specific use case.

In figure 9, you can see the 1-dimensional DFT of kernels calculated for a 64 long discrete signal. The kernels are categorized based on their size and sparsity: dense (0 % sparsity) and small (size: 3), dense and large (10), and sparse (30 % sparsity) and large (10) kernels. Only the positive frequencies are displayed, up to 31 ($N/2 - 1$ for $N = 64$), because, given that the kernels consist of real numbers, their frequency response is symmetric. It is observed that larger kernels typically exhibit greater variation in their DFT. This allows them to capture a more diverse range of frequencies, thus preserving

more details and reducing blurriness in the reconstructed image. However, larger kernels are computationally more expensive to process. To address this issue, we can use large sparse kernels, which have fewer non-zero elements but still achieve similar frequency variation. Sparse kernels can provide a good balance between computational efficiency and the ability to capture significant variations in frequency. Monte Carlo experiments, involving 3,000 randomly chosen kernels, reveal that the Total Variation (TV) was 1.4240 for small dense kernels, 5.3391 for large dense kernels, and 3.8236 for large sparse kernels. These results support our conjecture that larger kernels, whether dense or sparse, exhibit greater Total Variation in their Discrete Fourier Transform (DFT). However, this observation is based on empirical data derived from Monte Carlo simulations and requires further mathematical proof to establish its general validity.

In our approach, we explored not just any sparse kernels but specifically those where the non-zero coefficients form a curve. This heuristic is based on the idea that such arrangements might enhance the frequency variation captured by the kernel. The rationale is that the spatial arrangement of non-zero coefficients can influence the frequency response, potentially improving the effectiveness of the kernel in capturing desired frequency components.

3.2.5 Solution 4: Multiscale pyramid decomposition

In the same vein as solution 3, a multiscale pyramid decomposition can be employed to encourage the Variational Autoencoder (VAE) to capture higher-frequency components.

Multiscale pyramid The multiscale pyramid [12] decomposes an image into different scales of frequency, allowing for a hierarchical representation of image information.

The decomposition begins by expressing the image $I(x, y)$ in the frequency domain as $\hat{I}(u, v)$. The Steerable Pyramid employs a series of low-pass and high-pass filters to separate the image into different frequency bands.

The low-pass filter $H_L(u, v)$ is defined as:

$$H_L(r) = \begin{cases} 1 & \text{if } r \leq \frac{\pi}{4} \\ \cos\left(\frac{\pi}{2} \frac{\log\left(\frac{4r}{\pi}\right)}{\log 2}\right) & \text{if } \frac{\pi}{4} < r < \frac{\pi}{2} \\ 0 & \text{otherwise} \end{cases}$$

and the high-pass filter $H_H(u, v)$ is:

$$H_H(r) = \begin{cases} 1 & \text{if } r \geq \frac{\pi}{2} \\ \cos\left(\frac{\pi}{2} \frac{\log\left(\frac{2r}{\pi}\right)}{\log 2}\right) & \text{if } \frac{\pi}{4} < r < \frac{\pi}{2} \\ 0 & \text{otherwise} \end{cases}$$

In these expressions, r represents the radial distance from the origin in the frequency domain, calculated as $r = \sqrt{u^2 + v^2}$.

The smoothness of these filters is crucial to avoid aliasing and artifacts during decomposition and reconstruction. Smooth filters ensure that transitions between different frequency bands are gradual, leading to more accurate multiscale representations.

To decompose the image, these filters are applied to the frequency representation $\hat{I}(u, v)$:

$$I_{\text{low}}(u, v) = H_L(u, v) \cdot \hat{I}(u, v)$$
$$I_{\text{high}}(u, v) = H_H(u, v) \cdot \hat{I}(u, v)$$

The inverse Fourier transform is then applied to these filtered components to return to the spatial domain:

$$I_{\text{low}}(x, y) = \mathcal{F}^{-1}\{I_{\text{low}}(u, v)\}$$
$$I_{\text{high}}(x, y) = \mathcal{F}^{-1}\{I_{\text{high}}(u, v)\}$$

The pyramid has multiple layers, with each subsequent layer operating on the low-pass output of the previous layer. If the pyramid has n_{scale} layers, the image is recursively filtered, producing a set of high-pass filtered subbands and a final low-pass image.

The complete multiscale decomposition process can be summarized in the pseudo-algorithm 6.

Finally, the reconstruction of the image can be done by reversing the pyramid process, summing the contributions from each layer, starting with the lowest resolution and progressively adding higher-frequency details from the subbands.

Multiscale pyramid for VAE In the VAE framework, the multiscale pyramid decomposition is utilized to enhance the representation and reconstruction of images. The process is as follows:

1. Encoder: For each scale k, let $I_k(x, y)$ denote the image sub-band at scale k, where k ranges from 1 to K. These sub-bands $\{I_k(x, y)\}_{k=1}^K$ are obtained from the multiscale pyramid decomposition. The encoder network processes the entire set of sub-bands to produce a single latent representation z:

   $$z = \text{Encoder}\left(\{I_k(x, y)\}_{k=1}^K\right)$$

   Here, z is a compact latent representation capturing the essential features of the entire multiscale decomposition.

2. Decoder: The decoder network takes the latent representation z and reconstructs the frequency sub-bands $\{I'_k(x, y)\}_{k=1}^K$ for each scale k:

   $$\{I'_k(x, y)\}_{k=1}^K = \text{Decoder}(z)$$

   Each $I'_k(x, y)$ aims to closely approximate the original sub-band $I_k(x, y)$ from the multiscale decomposition.

3. Reconstruction Loss: To ensure accurate reconstruction, a reconstruction loss function is defined to measure the similarity between the original sub-bands $\{I_k(x, y)\}_{k=1}^K$ and the reconstructed sub-bands $\{I'_k(x, y)\}_{k=1}^K$. Higher weights λ_k are applied to high-frequency sub-bands to emphasize their importance:

   $$\text{Loss}_{\text{recon}} = \sum_{k=1}^{K} \lambda_k \cdot \text{ReconstructionLoss}(I_k(x, y), I'_k(x, y))$$

Algorithm 6 Multiscale Pyramid Decomposition

Input: Image $I(x, y)$, Number of scales K
Output: Pyramid of frequency subbands and final low-pass image

1. Compute the frequency representation of the image:

$$\hat{I}(u, v) = \mathcal{F}\{I(x, y)\}$$

2. Initialize an empty list for storing subbands:

$$\text{output} \leftarrow []$$

3. Set the initial low-resolution image:

$$\text{low\_res\_image} \leftarrow \hat{I}(u, v)$$

for each scale s from 0 to $K - 1$ **do**
   a. Compute the high-pass filtered component:

$$I_{\text{high}}(u, v) = H_H(u, v) \cdot \text{low\_res\_image}$$

   b. Append the high-pass subband $I_{\text{high}}(u, v)$ to the list:

$$\text{output} \leftarrow \text{output} \cup \{I_{\text{high}}(u, v)\}$$

   c. Update the low-resolution image for the next scale:

$$\text{low\_res\_image} \leftarrow H_L(u, v) \cdot \text{low\_res\_image}$$

end for
4. Append the final low-pass image to the list:

$$\text{output} \leftarrow \text{output} \cup \{\text{low\_res\_image}\}$$

Return: output (the pyramid consisting of high-pass subbands and final low-pass image)

where λ_k is a weight factor that increases with higher-frequency sub-bands to prioritize fine detail reconstruction.

4. Image Recombination Loss: After reconstructing the sub-bands $\{I'_k(x,y)\}_{k=1}^K$, they are recombined to form the final reconstructed image x'. The image recombination loss quantifies the difference between the original image $I(x,y)$ and the recomposed image x':
$$\text{Loss}_{\text{final}} = \text{ImageLoss}(I(x,y), x')$$
where $x' = \text{Recompose}\left(\{I'_k(x,y)\}_{k=1}^K\right)$ represents the image reconstructed from the high-pass and low-pass sub-bands. This loss term is optional, but further ensures that the recombination works.

5. KL Divergence Loss: In line with the VAE framework, the loss function includes the usual KL divergence term $\text{KL}\left(\hat{P}_{Z|X,\Phi}(\mathbf{z}|\mathbf{x}) \| P_{Z|\Theta}(\mathbf{z})\right)$

This approach ensures that the VAE effectively captures high-frequency details and accurately reconstructs images by integrating multiscale pyramid decomposition into both encoding and decoding processes.

3.3 Results

In this section, we present the results of the different VAEs for texture generation. The models compared include the standard VAE, Planar Flow VAE, VAE-GAN, Large Sparse Kernel Encoder, and Multiscale Pyramid Decomposition VAE. The experiments were run on a GPU with 256x256 image resolution, using a dataset consisting of 5,638 texture images. Note that the classification of some images as 'texture' may vary depending on the definition used; in this context, 'texture' is broadly defined to include integrated macro-textures. The textures are subdivided in 46 categories as 'bubbly' or 'cracked'. A latent space dimension of 256 was chosen experimentally to balance dimensionality reduction with image quality.

3.3.1 Model Setup and Parameters

Table 1 summarizes the number of parameters for each VAE architecture. Note that for the "Large Sparse Kernel" VAE, the actual number of meaningful parameters is the same as in the Standard VAE since during training, most of the kernel weights are set to zero (their gradients are also set to zero). Specifically, only 10% of the weights in a 10×10 kernel are non-zero, which is almost equivalent to having a fully active 3×3 kernel as used in the other VAEs.

The number of training epochs (up to 1,000+) and time (up to 5h) is not directly comparable across models as we pushed each model to its limits, with varying learning rates (from 1e-3 to 1e-6) and batch sizes (average: 128) depending on the GPU's capacity.

3.3.2 Texture Generation Results

Figure 10 illustrates the textures generated by each VAE architecture. Among them, the "Planar Flow" model produces the most realistic textures, followed closely by the "Multiscale Pyramid" model. Unfortunately, the "VAE-GAN" did not perform as expected.

Model	Number of Parameters	Total
Standard VAE	Encoder: 74m, Decoder: 74m	148m
Planar Flow VAE	Encoder: 74m, Decoder: 74m, Flow: 2.5k	148m
VAE-GAN	Encoder: 74m, Decoder: 74m, Discr: 25m	173m
Large Sparse Kernel	Encoder: 137m , Decoder: 74m	211m
Multiscale Pyramid	Encoder: 74m, Decoder: 74m	148m

Table 1: Number of Parameters for Different Models

While the discriminator was effective—assigning probabilities close to 1.00 for real images and near 0.00 for fake ones—it failed to back-propagate useful information to the VAE. It's worth noting that, although the standard VAE is quick to train, it reaches its performance limit rapidly. In contrast, the other models, while more challenging to train, ultimately, some of them yield better results. Note that for some technical error I cannot display the result for the "Large Sparse Kernel" VAE.

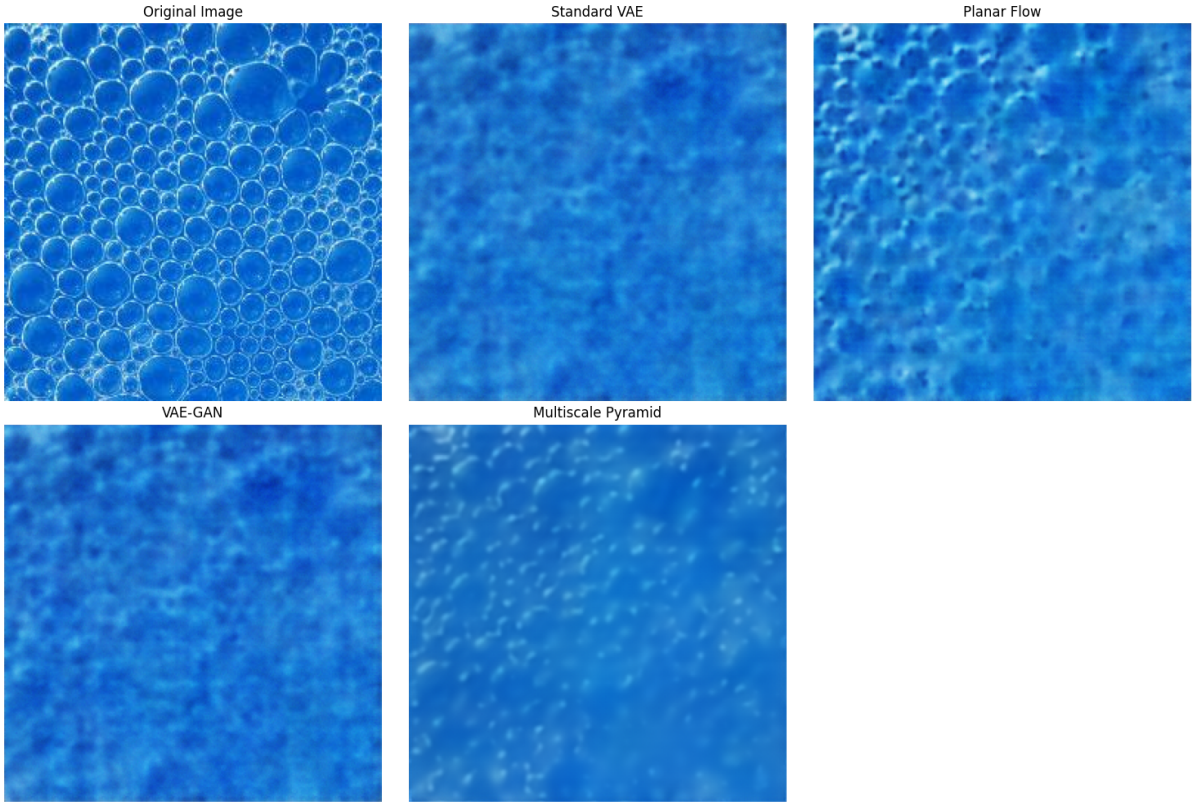

Figure 10: Generated textures using different VAE architectures

3.3.3 Comparison of High-Frequency Components and Reconstruction Loss

We compared the models based on their ability to capture high-frequency components and their average reconstruction loss. The results are summarized in Table 2.

The high-frequency energy quantifies the amount of energy present in the high-frequency components of the image. To compute this, we first convert the image to grayscale and then compute its 2D Fourier Transform. The power spectrum of the Fourier Transform

is used to measure the distribution of image energy across different frequencies. The high-frequency energy is derived by summing the power spectrum values that lie outside a central low-frequency region, which is masked based on a threshold relative to the maximum distance from the center of the spectrum.

The high-frequency energy ratio is computed as the ratio of the high-frequency energy to the total energy of the image. This ratio provides a normalized measure of how much of the image's energy is concentrated in high-frequency components.

Model	**High-Frequency Energy**	**High-Frequency Energy Ratio**	**Average Reconstruction Loss**
Standard VAE	205K	0.02%	0.0074
Planar Flow VAE	341K	0.03%	0.0052
VAE-GAN	218K	0.02%	0.0074
Large Sparse Kernel	NA	NA	0.0059
Multiscale Pyramid	13K	0.00%	0.0101

Table 2: Comparison of high-frequency components, high-frequency energy ratio, and average reconstruction loss across different VAE architectures.

Surprisingly, the Multiscale Pyramid model exhibit lower high-frequency energy compared to others, despite the expectation that these methods would enhance high-frequency details. This indicates that their performance in capturing fine texture details is not as effective as initially anticipated. Further investigation is required to understand the reasons behind this outcome.

3.3.4 Multiscale Pyramid Decomposition

The Multiscale Pyramid Decomposition VAE is further analyzed by decomposing the generated textures into different scales. As shown in Figure 11, the model effectively learns the low-frequency components of the textures. Some mid-frequency components are captured, but the high-frequency details are less accurately reproduced.

3.3.5 Latent Space Interpolation

Finally, we perform interpolation in the latent space between two textures. The first example, shown in Figure 12, demonstrates interpolation between textures with the same color but different patterns. The second example in Figure 13 shows interpolation between textures with different colors and patterns. These interpolations highlight the model's capability to smoothly transition between different textures in the latent space.

3.3.6 Code

All of the code related to the VAEs has been published on GitHub and can be found under this link here:.

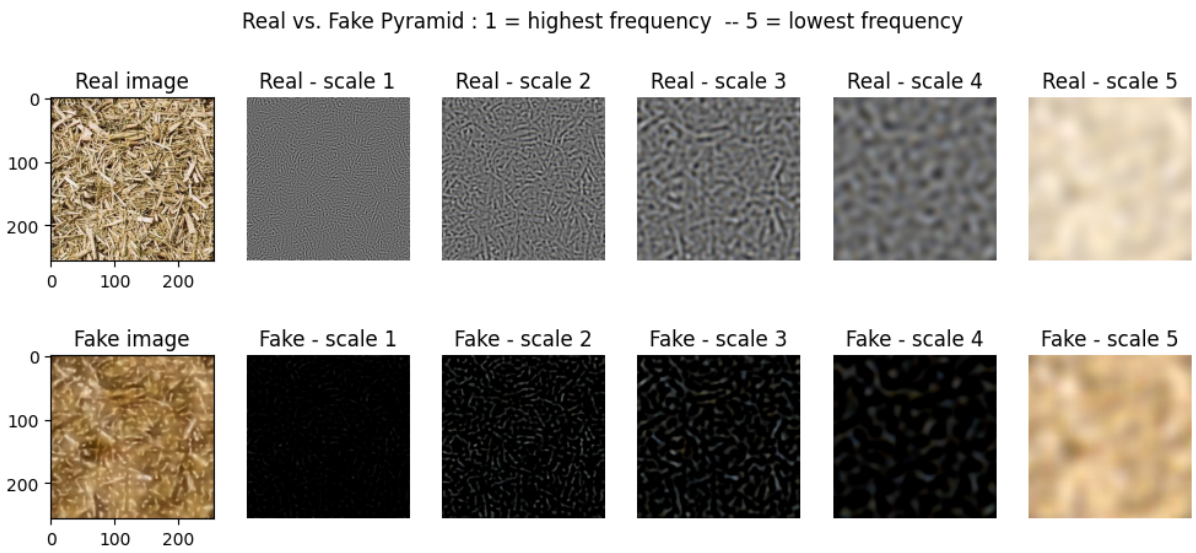

Figure 11: Multiscale decomposition of original vs generated textures

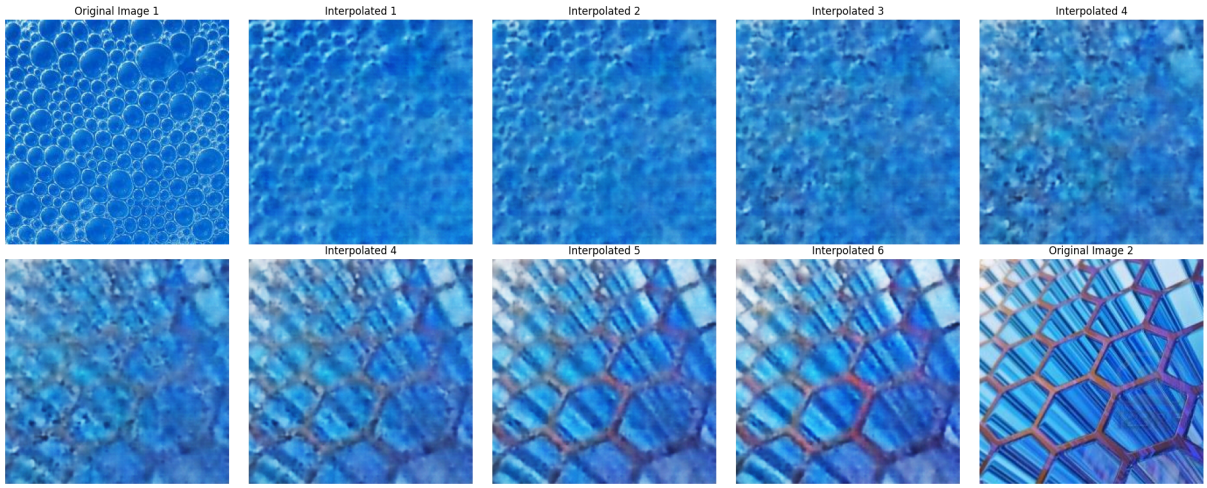

Figure 12: Latent space interpolation between textures with the same color but different patterns.

4 Conclusion

This report explores visual perception with a focus on perceptual scales, particularly how these scales can be derived and tested using various methods. The primary objective was to understand and evaluate perceptual scales for textures, which are broadly defined as images containing repetitive patterns. The perceptual scale quantifies the relationship between physical stimuli and their psychological representation. Our investigation spans two key areas: testing the assumptions behind Maximum Likelihood Difference Scaling (MLDS) and developing methods for generating and interpolating textures using Variational Autoencoders (VAEs).

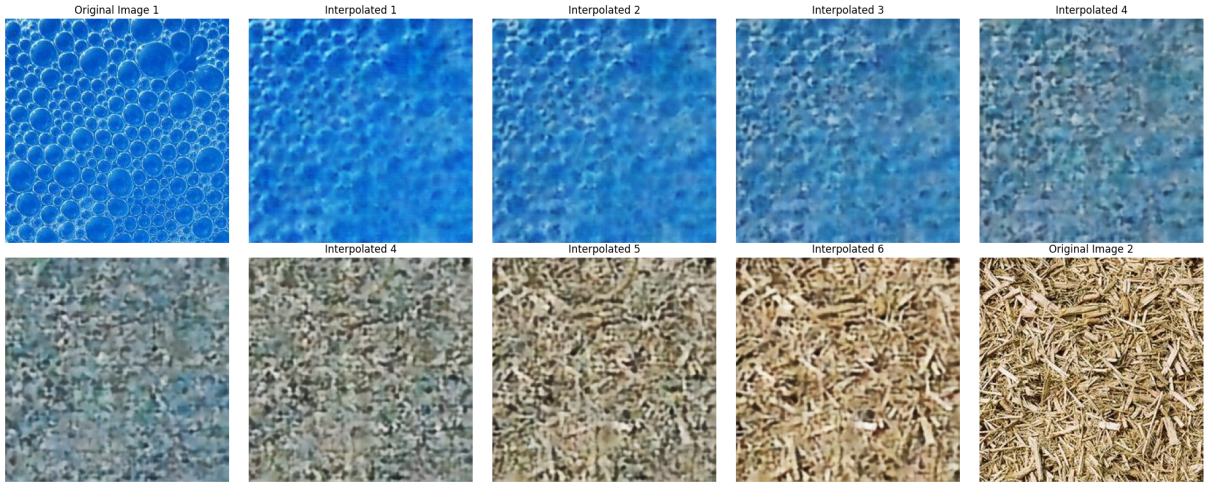

Figure 13: Latent space interpolation between textures with different colors and patterns.

4.1 Theory of MLDS

In Section 2, we examined the MLDS method for calculating perceptual scales. We recognized that the underlying assumption (H1) of MLDS—namely, that absolute values are irrelevant when comparing stimuli—may not always hold true. Empirical results demonstrated that MLDS underperforms when this assumption does not hold. We proposed a method to empirically test these assumptions. The next step involves applying this proposed method experimentally to a known perceptual scale. Additionally, further research could explore the potential of adapting MLDS for alternative assumptions, which may enhance its applicability in cases where H1 does not apply.

4.2 Texture Generation with VAEs

The Section 3 of the report focused on texture synthesis using VAEs. VAEs are effective generators as they drastically reduce dimensionality, create meaningful latent spaces, and model textures as probabilistic distributions, aligning with the Bayesian framework of visual perception. However, a significant challenge identified was the blurriness of generated textures. To address this issue, we explored several strategies:

- Modifying the VAE architecture with large sparse kernels to capture a greater variety of frequencies, thereby improving image sharpness.
- Enhancing posterior complexity using planar flow to improve the Evidence Lower Bound (ELBO) and address the high variance inherent to classic VAEs.
- Incorporating discriminator feedback to mitigate blurriness and enhance overall image quality.
- Decomposing images into frequency components to better train higher frequencies and reduce blurriness.

These results are preliminary and were achieved within a limited time-frame. To fully harness the potential of these strategies, further fine-tuning and adjustments would be necessary.

4.3 Evaluation and Future Directions

The results from these methods indicate promising improvements in texture quality. However, textures should not be evaluated solely based on pixel-wise Mean Squared Error (MSE) as the loss metric. Since textures are stationary random fields, loss metrics must be invariant to transformations such as rotation and translation to accurately reflect their nature. Testing with Generative Adversarial Networks (GANs) was attempted but did not meet expectations.

A thorough analysis of each method's efficiency, including time complexity and computational resource requirements, is crucial for determining the most effective approach to generating high-quality textures. Combining some of the tested methods could also be explored. Furthermore, integrating state-of-the-art generative models, such as diffusion models alongside VAEs, may yield even better results.

Additional validation is needed to ensure that the generated textures are suitable for MLDS experiments. It is essential to assess whether these textures meet the experimental requirements for perceptual scale derivation.

Once the textures are optimized, they can be applied to MLDS experiments. In this context, using interpolation between textures as a physical parameter could enhance the study of perceptual scales. Given the high dimensionality of textures, interpolation on a plane between multiple textures could allow for a more nuanced analysis of perceptual scales.

In summary, while significant progress has been made in generating and evaluating textures, ongoing refinement and validation are essential. The integration of new models and methods will pave the way for more precise and meaningful studies of perceptual scales.

Bibliography

References

[1] Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *ArXiv*, abs/1701.04862, 2017.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017. cite arxiv:1701.07875.

[3] Gustav Bredell, Kyriakos Flouris, Krishna Chaitanya, Ertunc Erdil, and Ender Konukoglu. Explicitly minimizing the blur error of variational autoencoders, 2023.

[4] Jiu Ding and Aihui Zhou. Eigenvalues of rank-one updated matrices with some applications. *Applied Mathematics Letters*, 20(12):1223–1226, 2007.

[5] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. 05 2015.

[6] B. Julesz. Visual pattern discrimination. *IRE Transactions on Information Theory*, 8(2):84–92, 1962.

[7] David C. Knill and Alexandre Pouget. The bayesian brain: the role of uncertainty in neural coding and computation. *Trends in Neurosciences*, 27(12):712–719, 2004.

[8] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1558–1566, New York, New York, USA, 20–22 Jun 2016. PMLR.

[9] Laurence T Maloney and Joong Nam Yang. Maximum likelihood difference scaling. *Journal of Vision*, 3(8):5–5, 2003.

[10] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[11] L. Ning. Smooth interpolation of covariance matrices and brain network estimation: Part ii. *IEEE Transactions on Automatic Control*, 65(5):1901–1910, May 2020.

[12] E. P. Simoncelli and W. T. Freeman. The steerable pyramid: a flexible architecture for multi-scale derivative computation. In *Proceedings of the 1995 International Conference on Image Processing (Vol. 3)-Volume 3 - Volume 3*, ICIP '95, page 3444, USA, 1995. IEEE Computer Society.

[13] Jonathan Vacher and Thibaud Briand. The Portilla-Simoncelli Texture Model: towards Understanding the Early Visual Cortex. *Image Processing On Line*, 11:170–211, 2021. `https://doi.org/10.5201/ipol.2021.324`.

[14] Yufeng Zhang, Jialu Pan, Li Ken Li, Wanwei Liu, Zhenbang Chen, Xinwang Liu, and J Wang. On the properties of kullback-leibler divergence between multivariate gaussian distributions. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 58152–58165. Curran Associates, Inc., 2023.

Glossary

Equation of Perception A mathematical model describing how perceived differences between stimuli are influenced by internal factors, including noise.

Frequency In image processing, this refers to the rate of change in pixel intensity values. High frequencies correspond to rapid changes and fine details, while low frequencies correspond to gradual changes and overall image structure.

GAN (Generative Adversarial Network) A framework where two neural networks, a generator and a discriminator, are trained simultaneously. The generator creates data, while the discriminator evaluates its realism, driving the generator to produce increasingly realistic samples.

Latent Space A compressed, lower-dimensional space learned by a generative model, representing the essential features of data. This space allows for the generation and manipulation of data by capturing its underlying structure in a more manageable form.

MLDS (Maximum Likelihood Difference Scaling) A method used to derive a perceptual scale by comparing perceived differences between stimuli. It involves measuring how differences between pairs of stimuli are judged, and adjusting scale values to best fit these judgments.

Perceptual Scale A representation of how differences between physical stimuli correspond to perceived differences. This scale transforms physical attributes of stimuli into a psychological representation that reflects how differences are perceived by an observer.

Psychometric Function A curve that illustrates the relationship between stimulus parameters and the probability of a particular response from an observer. It is used to measure perceptual sensitivity.

Texture An image characterized by repetitive patterns or structures. It is often modeled as a realization of a stationary random field, where its statistical properties are invariant under translations in the image plane.

Visual Perception The process by which the brain interprets and makes sense of visual stimuli. This involves detecting and processing light through the eyes, transmitting visual information via the optic nerve to the visual cortex, and interpreting this information to form a coherent internal representation of the environment.

Visual Stimuli Objects or patterns presented to the visual system for study, used to investigate how visual perception operates and to understand various aspects of visual processing and interpretation.

VAE (Variational Autoencoder) A generative model that learns to encode data into a lower-dimensional latent space and decode it back, facilitating tasks such as data generation and feature extraction by capturing complex distributions in a compressed form.

Appendices

A MLDS ALgorithm

Listing 1: MLDS Algorithm Implementation

```
def negloglikelihood(params, nb_indices, quadruples, results, nb_trials):
    """
    Calculate the negative log-likelihood for a given set of parameters.

    Parameters:
    - params: List of parameters to optimize, i.e the value of the perceptual scale function
    and the std of the noise
    - nb_indices: Number of indices used to map quadruples.
    - quadruples: A list of quadruples representing the stimuli.
    - results: List of trial results.
    - nb_trials: Number of trials in the dataset.

    Returns:
    - ll: The negative log-likelihood value.
    """
    w = [0] + list(params[:-1]) + [1]  # Add fixed boundary values to weights
    sigma = params[-1]  # Extract sigma parameter
    ll = 0

    for t in range(nb_trials):
        for i, quad in enumerate(quadruples):
            d = (w[int(quad[1] * nb_indices)] - w[int(quad[0] * nb_indices)]) \
                - (w[int(quad[3] * nb_indices)] - w[int(quad[2] * nb_indices)])

            ll += np.log(norm.cdf(d / sigma) ** results[t][i]) \
                  + np.log((1 - norm.cdf(d / sigma)) ** (1 - results[t][i]))

            if i == len(quadruples) - 1:
                print(f'Log-likelihood at trial {t+1}/{nb_trials}: {ll}')

    return -ll

def ll_fit(initial_guess, negloglikelihood, nb_indices, quadruples, results, nb_trials):
    """
    Perform optimization to minimize the negative log-likelihood.

    Parameters:
    - initial_guess: Initial guess for the parameters to be optimized.
    - negloglikelihood: Function to calculate the negative log-likelihood.
    - nb_indices: Number of indices used to map quadruples.
    - quadruples: A list of quadruples representing the stimuli.
    - results: List of trial results (R_t).
    - nb_trials: Number of trials in the dataset. (T)

    Returns:
    - optimal_solution: The optimized values of the perceptual scale.
    - optimal_value: The corresponding log-likelihood.
    """
    constraints = [
        {'type': 'ineq', 'fun': lambda x: x},  # Ensure all weights are >= 0
        {'type': 'ineq', 'fun': lambda x: 1 - x},  # Ensure all weights are <= 1
    ]

    for i in range(len(initial_guess) - 2):
        constraints.append({'type': 'ineq', 'fun': lambda x, i=i: x[i + 1] - x[i]})
# Ensure weights are increasing

    opt = minimize(
        lambda params: negloglikelihood(params, nb_indices, quadruples, results, nb_trials),
        initial_guess, method='SLSQP', constraints=constraints, options={'maxiter': 100}
    )

    optimal_solution = opt.x
    optimal_value = opt.fun

    return optimal_solution, optimal_value
```