Master en Ciencia de Datos

Trabajo Fin de Master

Parametric Deep Learning: Introducing statistical and biological knowledge via functional forms

Autor: Jorge Vila Tomás

Tutores: Jesús Malo López

Valero Laparra Pérez-Muelas

Julio 2023

Trabajo Fin de Master

Parametric Deep Learning: Introducing statistical and biological knowledge via functional forms

Autor: Jorge Vila Tomás

Tutores: Jesús Malo López

Valero Laparra Pérez-Muelas

TRIBUNAL

PRESIDENTE/A:

VOCAL 1:

VOCAL 2:

FECHA DE DEFENSA:

CALIFICACIÓN:

Declaración de autoría:

Yo, Jorge Vila Tomás, declaro la autoría del Trabajo Fin de Master titulado "Parametric Deep Learning: Introducing statistical and biological knowledge via functional forms" y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.

Valencia, October 31, 2023

Fdo: Jorge Vila Tomás

Resumen:

Este trabajo toma inspiración de la estadística y la biología para definir un conjunto de funciones paramétricas (Gaussiana, Gabor, Centro Periferia y Normalización Divisiva) y las introduce en el campo de trabajo del aprendizaje profundo para entrenar sus parámetros. Estas funciones se analizan en dos "problemas de juguete" para comprobar la correcta implementación y explorar las peculiaridades de optimizar este tipo de funciones frente al enfoque usual en el aprendizaje profundo donde se dejan todos los parámetros libres sin restricciones. Para terminar, se aplican estas formas funcionales a dos problemas reales: clasificación y calidad de imagen, donde superan a los enfoques no paramétricos obteniendo mejores resultados y generalizando mejor con una menor cantidad de parámetros entrenables y, al mismo tiempo, mostrando que los modelos paramétricos pueden ser entrenados con tasas de entrenamiento (*learning rate*) más altas para obtener mejores resultados de forma más rápida que con los modelos no paramétricos.

Abstract:

This work takes inspiration from statistics and biological findings to define a set of parametric functions (Gaussian, Gabor, Center Surround and Divisive Normalization) and implements them into the conventional deep learning framework. These functions are analyzed in two different toy problems to assess their correctness and explore the peculiarities of optimizing them versus optimizing non-parametric models as is the norm in the deep learning field. Finally, these functional forms are applied to two real world problems: classification and image quality, where they outperform non-parametric approaches by obtaining better performances, and generalization with fewer number of trainable parameters while also showing that parametric models can be trained with higher learning rates to obtain better results faster than non-parametric models.

Resum:

Aquest treball pren inspiració de l'estadística i la biologia per a definir un conjunt de funcions paramètriques (Gaussiana, Gabor, Centre Perifèria i Normalització Divisiva) i les introdueix en el camp de treball de l'aprenentatge profund per a entrenar els seus paràmetres. Aquestes funcions s'analitzen en dos "problemes de joguet" per a comprovar la correcta implementació i explorar les peculiaritats d'optimitzar aquest tipus de funcions enfront de l'enfocament usual en l'aprenentatge profund on es deixen tots els paràmetres lliures sense restriccions. Per a acabar, s'apliquen aquestes formes funcionals a dos problemes reals: classificació i qualitat d'imatge, on superen als plantejaments no paramètrics obtenint millors resultats i generalitzant millor amb una menor quantitat de paràmetres entrenables i, al mateix temps, mostrant que els models paramètrics poden ser entrenats amb taxes d'entrenament (*learning rate*) més altes per a obtindre millors resultats de forma més ràpida que amb els models no paramètrics.

Agradecimientos:

This work was supported in part by MICIIN/FEDER/UE under Grant PID2020-118071GB-I00 and PDC2021-121522-C21, in part by Spanish MIU under Grant FPU21/02256 and in part by Generalitat Valenciana under Projects GV/2021/074, CIPROM/2021/056 and CIAPOT/2021/9. Some computer resources were provided by Artemisa, funded by the European Union ERDF and Comunitat Valenciana as well as the technical support provided by the Instituto de Física Corpuscular, IFIC (CSIC-UV).

Contents

Chapter 1

Introduction

The breakthrough of convolutional neural networks started with [Lecun et al., 1998] and moved rapidly as its impressive performance reignited the spark and interest of neural networks, giving rise to big jumps in performance in the competitive ImageNet contest with the appearance of networks like the VGG [Simonyan and Zisserman, 2015], ResNet [He et al., 2015] and Inception [Szegedy et al., 2014]. Even though deep neural networks were known for overfitting the training data distribution [Lecun et al., 1998], the increase in computing power and available data started a golden age for convolutional neural networks, which quickly moved to the top spots in all the vision-related challenges' leaderboards.

The usual approach to training convolutional neural networks (CNNs) consists of defining a set of convolutional kernels of predefined spatial size (kx, ky) whose pixels are all independently-trainable. This means that, for a kernel $k \in \mathcal{R}^{kx,ky}$, there are $kx \times ky$ trainable weights. In the most common setting for images, the input to a convolutional layer L would have 3 dimensions (height, width and channels): $I \in \mathcal{R}^{H,W,C}$ and, normally, each layer of a CNN has several kernels, usually called *features*, F, resulting in an output of shape $O \in \mathcal{R}^{H,W,F}$ if considering the appropriate padding of the input so that the spatial dimensions are not changed after the convolution operation. In this setting, the complete kernel of L would have dimensions $k \in \mathrm{R}^{kx,ky,C,F}$, amounting to a total number of $kx \times ky \times C \times F$ trainable weights. One can see the number of trainable weights quickly growing with the spatial size of the kernels and, considering that the number of features also grows with the depth of the network, deeper layers could have a significant number of parameters. It is so the case if we look at the number of parameters from some ImageNet winning models in Table 1.1, we see that all of them are surpassing a million parameters. And they don't even have filters bigger than $7x7$!

The number of parameters is (and will be) a problem for researchers and practitioners that do not have access to the computing resources needed to train the ever-growing models that keep arising. Besides this, with bigger models, their interpretability becomes

Table 1.1: Parameter count of some of the more well-known models that participated in the ImageNet challenge. All of them correspond to their smallest versions.

Model	# Parameters
VGG16	14.780.352
ResNet18	10.998.784
Google LeNet (Inception)	5.389.360

harder and their generalization may suffer as well because having more parameters means the model would be more prone to memorizing the training data. In [Olah et al., 2017], a method to obtain and visualize the features of interest for a network is presented. While this work focuses mainly on higher-level features that are appealing and can be visually interpreted, the intermediate-level features of the filters are harder to interpret, as the filters usually learn odd forms and shapes from the training data. However, earlier layers of artificial neural networks are known to learn, easy-to-characterize Gabor-like filters, edge detectors, etc [Krizhevsky et al., 2012], and this is often praised as a way of showing that CNNs are "learning what they should learn" but, **if that is what we want them to learn, why aren't we telling them straight away?**

Examples of prior knowledge of what "CNNs devoted to vision should learn" are given in Fig1.1. By looking at the biological visual system, it has been found that it displays center-surround receptive fields in the retina-LGN [Enroth-Cugell and Robson, 1966] and Gabor-like filters in V1 cortex [Hubel and Wiesel, 1962, Ringach, 2002] (Figure 1.1 top-left). However, this is not only a biological finding. Applying statistical-based tools like second-order (PCA) and higher-order redundancy reduction techniques leads to local Fourier-like and Gabor-like filters [Hyvärinen and Oja, 2000, Olshausen and Field, 1996] (Figure 1.1 top-right). Therefore, it is not surprising, but encouraging and explainable, that networks devoted to vision developed Gabor-like filters [Krizhevsky et al., 2012] (Figure 1.1 bottom).

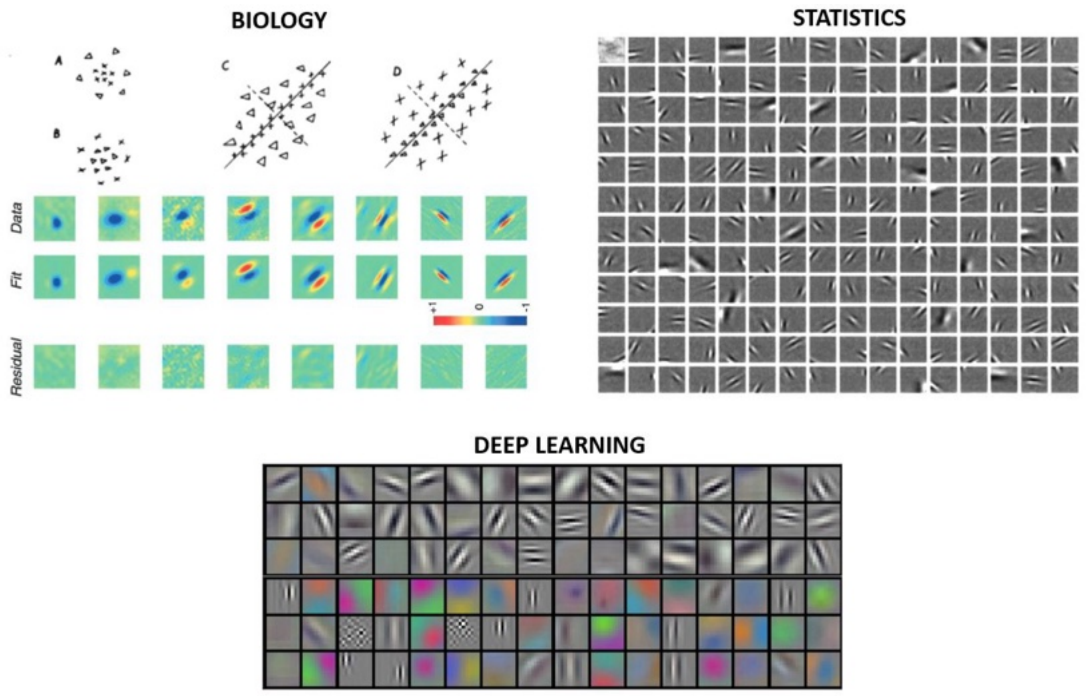


Figure 1.1: Consistent image representations that emerge in Biology, Statistics, and Deep Learning. The *top-left panel* represents the drawings by the Nobel Price laureates [Hubel and Wiesel, 1962] of different receptive fields of cells along the visual pathway: the center-surround LGN neurons, and the Gabor-like neurons in the visual cortex V1. Moreover, it shows actual measurements of the V1 receptive fields (data) using current reverse-correlation techniques and fits to Gabor functions [Ringach, 2002]. The *top-right panel* shows the Gabor-like representation that emerges to capture the independent components of natural images [Olshausen and Field, 1996]. The *bottom panel* shows the Gabor-like representation that emerges in the early layers of Alexnet optimized for image classification [Krizhevsky et al., 2012].

In order to tackle these issues (number of parameters, explainability, and preconditioning), scientists became motivated (and inspired by statistics and biology) to constraint the kernels on the earlier convolutional layers to a specific functional form (i.e. opponent color channels, local-frequency sensors, and Gabor functions), showing faster and better training [Alekseev and Bobe, 2019], better generalization to out of the sample data [Evans et al., 2022], or enhancing robustness to adversarial attacks [Pérez et al., 2020]. It's worth noting that implementing specific functional forms (like Gaussians) allows the consideration of more sophisticated (non-linear) bio-inspired operations like *Divisive Normalization* [Carandini and Heeger, 1994] into neural networks [Veerabadran et al., 2021], which also have appealing statistical effects [Schwartz and Simoncelli, 2001].

This work is devoted to the study of these parametric characterizations of linear and nonlinear operations in deep neural networks, explaining their advantages in interpretability and generalization capabilities.

Chapter 2

Objectives

We can separate the objectives of this work between two technical and a two-sided scientific objective. From the technical side we have:

1. Be able to impose a parametric form to a certain neural network's weights and optimize its parameters through gradient descent.
2. Test our implementation on a set of tasks with a known solution to assert their correct functioning and inspect their training dynamics.

While the scientific objective is:

3. Apply our restricted weights to two applications by building a statistically and biologically inspired network whose layers have a specific (imposed) meaning:
    (a) Classification.
    (b) Image Quality Assessment.

The scientific objectives use the result of our two technical objectives and take knowledge from the statistics of natural images and biological neural processing to solve both tasks.

Chapter 3

Review of the field's landscape

This Chapter consists of a brief overview of the terminology and basic concepts used in the artificial neural networks' field followed by a review of different approaches that take inspiration from statistics and biology to modify the training procedure employed to train the models, the tasks and data used to this effect, or the models themselves. After identifying interesting functional forms that add this statistical and biological knowledge, we will go through different approaches that have been taken to include them into artificial neural networks.

3.1 Neural Networks: Basic Concepts

The field of artificial neural networks is in constant movement so, before diving deeper into the particularities of this work, we thought of giving the reader a very brief introduction to some jargon and concepts that will be appearing in the following pages so as to lift everyone up to the same level and facilitate the comprehension of our work. We will be going through the concept of gradient descent, the key component that allows practitioners to train from very small to huge models, we will as well cover the basic operation to be used in our models: the convolution, and we'll finish with a note on activation functions and what we refer to when "activating a function".

3.1.1 Gradient Descent

Even in high school, when teaching about simple optimization methods, the use of derivatives makes an appearance. By definition, the derivative of an univariate function $f(x)$, $f'(x)$, points towards the direction of maximum change meaning that, when x is taken to be the minimum or maximum of the function, $f'(x) = 0$. Artificial neural networks are no more than functions, N_θ, depending on a set of parameters θ that are applied to some inputs x to produce an output $y = N_\theta(x)$. How could we proceed to find the set of parameters $\hat{\theta}$ that minimize (or maximize, depending on the problem at hand) a given loss function $\mathcal{L}$? With derivatives! Specifically, we will calculate the derivative of $\mathcal{L}$ with respect to the parameters θ, $\nabla_\theta \mathcal{L}$, to obtain the direction in which we want to move in order to reach the parameters $\hat{\theta}$ that optimize our function $\mathcal{L}$. This multivariate derivative is also known as the gradient of $\mathcal{L}$ with respect to the parameters θ. Keep in mind that the gradient points towards the maximum so, if we want to minimize the function, we will have to move in the opposite direction.

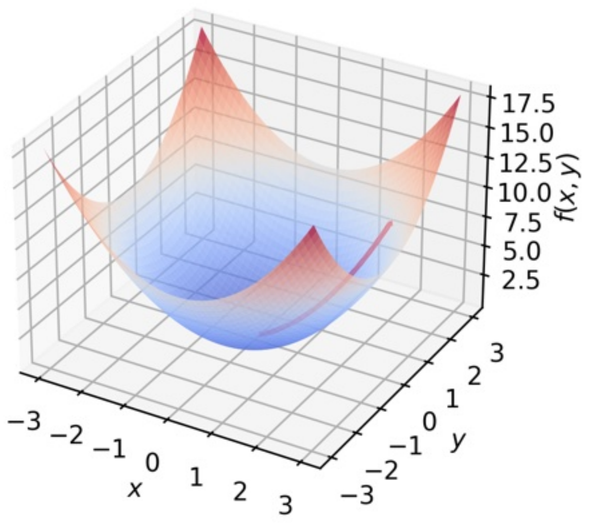


Figure 3.1: Minimization of the function $f(x,y) = x^2 + y^2$ with gradient descent for demonstration purposes.

Having laid everything out, we can take an iterative approach in which we initialize θ to some (usually random) values, calculate $\nabla_\theta \mathcal{L}$ and change the parameters in its opposite direction, leading to Eq. 3.1, the expression that models the training process of every artificial neural network, where the hyperparameter α is called "learning rate" and controls the step size at each iteration. Note that $0 < \alpha < 1$ or the training will diverge. Figure 3.1 shows a trivial example minimizing the function $f(x,y) = x^2 + y^2$ with gradient descent.

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L} \tag{3.1}$$

Originally, it was thought that the gradient should be computed as the mean gradient for the whole dataset but it would take a good amount of time to perform a parameter update, making the training process very slow. The opposite procedure was also put to the test: updating the parameters with the gradient calculated from a single sample. This led to faster training but the models were prone to overfit. Finally, the community settled on a middle point: split the full data set into subsets (called "batches") and perform the updates with the mean gradient of all the samples in a batch. This received the name of *stochastic gradient descent* and *was found to converge much faster than the true gradient* while finding *solutions that are more robust* [Lecun et al., 1998].

3.1.2 Convolution

In this work, we are going to be focused on deep convolutional neural networks, which are mainly based on learning a set of filters to convolve with an input to solve a specific task. The convolution operation can be defined both for continuous and discrete signals. However, due to the discrete nature of the images, when saying "convolution", we will be referring to the discrete convolution. The mathematical definition of the 2D convolution is stated in Eq. 3.2:

$$(f * g)[n,m] = \sum_{i=-k_x}^{k_x} \sum_{j=-k_y}^{k_y} f[n-i, m-j]g[i,j] \tag{3.2}$$

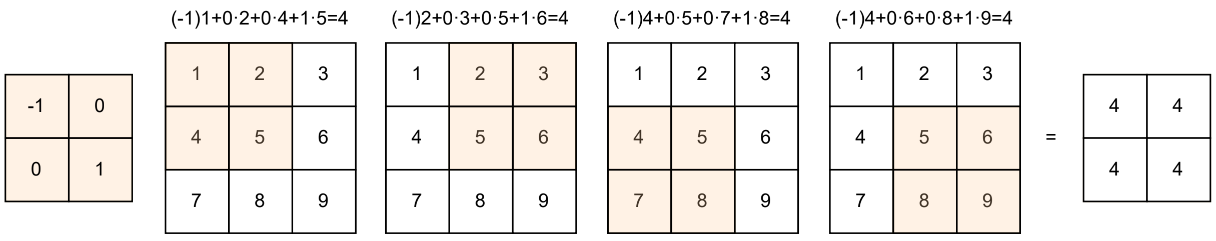


Figure 3.2: Diagram showing the convolution operation of a 3x3 matrix (in white) with a 2x2 filter (in light orange) using a stride of 1 and no padding. If not using padding, the output image is of shape $(3 - \frac{2}{2}, 3 - \frac{2}{2})$.

We will apply this definition to the correlation between an image and a filter (or a set of filters when considering more than one), so f will be a (I_x, I_y) matrix, and g will be a (k_x, k_y) matrix.

The mathematical definition in Eq. 3.2 can be a little bit intimidating at first, but what it is telling us to do is actually very simple (shown in Figure 3.2): move the filter, g, across the image and do a weighted sum of the filter and the "overlayed" image values. Strictly speaking, this operation could be called *cross-correlation* between the image and the filter, but everyone in the field refers to it as convolution. There are some details to be considered when performing this operation: strides, padding, dilation, grouping, etc, but we will only explain briefly the first two. Normally, it's advised to utilize odd-size filters so that they have "a center", but, when applying the (default) convolution, the center doesn't pass for every pixel, resulting in an output image of smaller size than the input. To fix this issue, a common practice is to add a border to the image so that the center of the filter passes over every pixel of the image and the spatial size is preserved between the input and output. This is called *padding* the input image. Incidentally, in Eq. 3.2 we have considered that the filter is displaced 1 pixel at each step, but this step size, the *stride*, can be chosen freely, keeping in mind that it will also affect the spatial size of the outputs. Considering all of this, the output size, (O_x, O_y), can be calculated with Eq. 3.3, where (p_x, p_y) and (s_x, s_y) are the padding and the strides in each dimension respectively:

$$(O_x, O_y) = \left(\frac{I_x - \frac{k_x}{2} + p_x}{s_x}, \frac{I_y - \frac{k_y}{2} + p_y}{s_y} \right) \tag{3.3}$$

3.1.3 Activation functions

It is known that artificial neural networks are based on stacking a set of linear and non-linear transformations in order to minimize a specific loss function with a given set of data. The two most common linear operations found, f, are matrix products and convolutions (which can even be defined as a matrix product). After applying any of those to a given input x, its output, $y = f(x)$, is "activated" with a non-linear function a to obtain its "activation", $z = a(y) = a(f(x))$. Normally, the sequential application of f and a is considered to be a "layer", and a is known as the activation function of the layer.

The most common activation functions are usually rectified or saturating functions. Three of the more conventionally seen are ReLU [Fukushima, 1975] (Eq. 3.4),

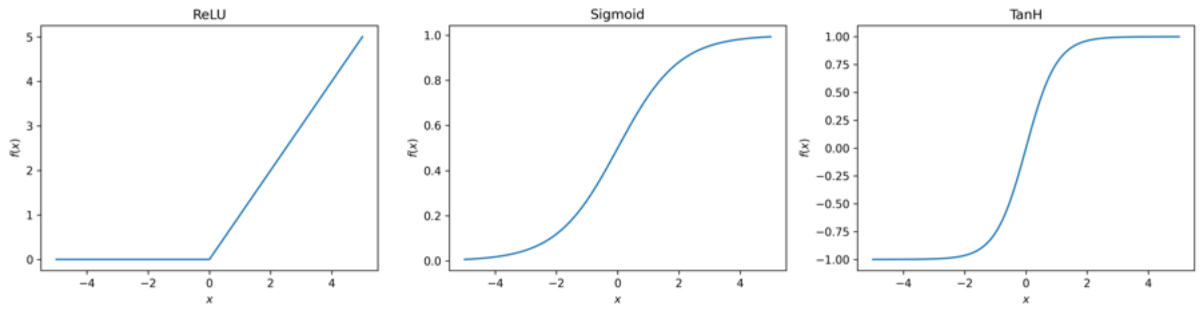


Figure 3.3: Three of the most common activation functions used in deep neural networks. ReLU is of the class of rectified functions while sigmoid and tanh pertain to the saturating functions family.

$$f(x) = \begin{cases} x, & \text{x} > 0 \\ 0, & x \leq 0 \end{cases} \tag{3.4}$$

sigmoid [Narayan, 1997] (Eq. 3.5),

$$f(x) = \frac{1}{1 - e^{-x}} \tag{3.5}$$

and tanh [LeCun et al., 2012] (Eq. 3.6),

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.6}$$

all shown in Figure 3.3.

3.2 Bio-inspired neural networks

Since their inception, neural networks have been considered to be inspired by the human brain but, actually, this inspiration is merely seen on *the form of the computation performed* [Bengio et al., 2016]. More specific features such as feedback connections, specific activation functions [Teo and Heeger, 1994] or even particular learning algorithms [Bengio et al., 2016, Hinton, 2022] have yet to be introduced properly in the field. What makes neural networks interesting is that, even though there's much work to do yet, we can still find correlations between their behavior and human-brain responses [Gomez-Villa et al., 2020, Li et al., 2022, Vila-Tomás et al., 2022b, Vila-Tomás et al., 2022a].

In this Section we will explore some of the approaches that have been taken towards increasing the biological inspiration in different areas of deep learning, starting from the training procedure, followed by the task and data used to train the models, and finishing with specific models or architectures that have been proposed with a more-than-the-norm focus on imposing statistical properties in image representations and increased consistency with biology, keeping in mind that we will focus in the model part within this work.

3.2.1 Training

One of the first critiques conventional deep learning gets with regard to its (dis)similarity to the brain is that backpropagation [Rumelhart et al., 1986] (the algorithm used to optimize the weights in an artificial neural network) is not actually happening in the brain [Markram and Sakmann, 1995, Gerstner et al., 1996]. In an effort to suggest a more brain-like training procedure, [Bengio et al., 2016] propose an interpretation of the spiking dynamics (Spike-Timing-Dependent Plasticity or STDP) that is known to happen in the human brain, which is related to the fact that real neurons may only fire binary responses [DeWeese et al., 2003]. Briefly, they propose to model STDP with Eq. 3.7, where ΔW_{ij} indicates the average weights change, S_i the pre-synaptic spike from neuron i, and $\dot{V}_j$ represents the temporal derivative of the post-synaptic voltage potential of neuron j.

$$\Delta W_{ij} \propto S_i \dot{V}_j \tag{3.7}$$

If we assume that ΔV_j may correspond to an improvement on some objective function J, STDP can be interpreted as *an approximation of stochastic gradient descent in the objective function.* In the end, what we see is that it might be easier to come up with an interpretation of backpropagation that satisfies its oppositors than proposing a new algorithm to train deep neural networks which can produce results as good as can be obtained with backpropagation.

As said in [Hinton, 2022], *backpropagation remains implausible despite considerable effort to invent ways in which it could be implemented by real neurons.* In an effort to overcome backpropagation, an alternative method is proposed in [Hinton, 2022]: the Forward-Forward algorithm. Inspired by Boltzmann machines [Rumelhart et al., 1986] and Noise Contrastive Estimation [Gutmann and Hyvärinen, 2010], this algorithm aims to replace the forward-backward passes of backpropagation with two forward passes (called positive and negative passes) operating in the same way but on different data: the positive pass utilizes real examples to adjust the model's weights to maximize a *goodness* function, while the negative pass operates on "negative" examples and modifies the weights to minimize their goodness. This goodness function can be of different types but, in the paper, they test both the positive and negative quadratic sum of the activations of the layers. Their conclusions are that *the Forward-Forward algorithm is somewhat slower than backpropagation and does not generalize quite as well on several toy problems*, so they don't see it as a replacement for backpropagation yet but, still, it is suggested that it *may be superior to backpropagation as a model of learning in cortex.*

Besides modifying the base training algorithm completely, in [Liu et al., 2023] a modification to usual backpropagation that is inspired by the brain's modularity is proposed, Brain-Inspired Modular Training, with the objective to improve the interpretability of artificial neural networks. In this work, each neuron is embeded in a geometric space and a term penalizing distant conections between neurons is introduced in the loss function (Eq. 3.8),

$$\mathcal{L} = L + l^w, \quad l^w = \sum_i^L \sum_j^{n_i} \sum_k^{n_{i+1}} d_{ijk} \left| w_{ijk} \right| \tag{3.8}$$

where L represents the conventional loss function for a given problem and l^w is a sum of the distances between all the neurons, d_{ijk}, weighted by the modulus of the weight

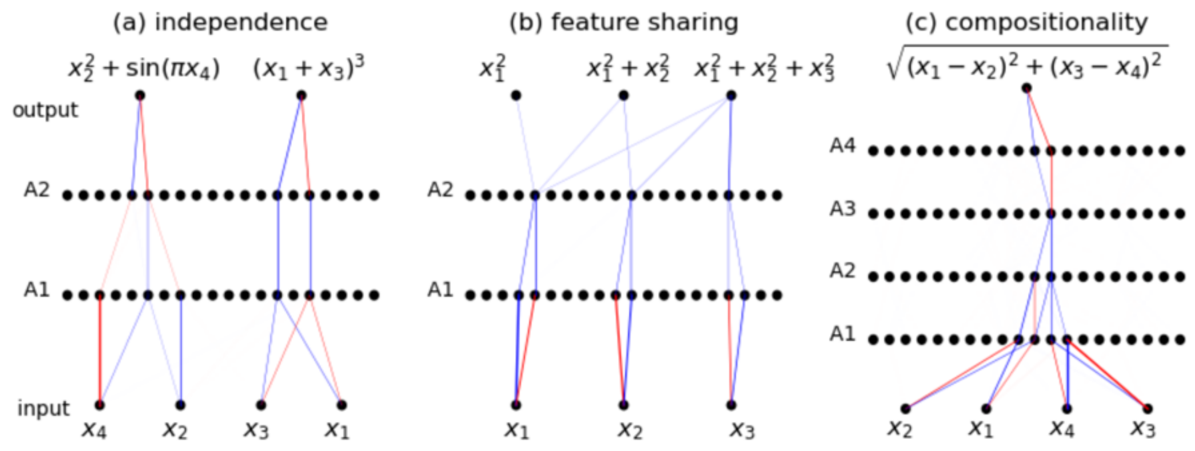


Figure 3.4: Conectivity graphs of different networks trained with Brain-Inspired Modular Training (BIMT) to solve a series of simple regression problems. It can be seen that the connections organize to show different behaviors such as independence (a), feature sharing (b), and compositionality (c). The main takeaway of these examples is that the neurons that need to work together to solve the task, are rearranged so that they are together. Extracted from [Liu et al., 2023].

that connects them, $|w_{ijk}|$. By doing so, they show that the networks become more easily interpretable and their calculations can be visually analyzed as they show in Figure 3.4.

3.2.2 Tasks and Data

Neural networks are known to learn the statistics of the data used to train them [Hepburn et al., 2022] and this, in fact, is very similar to how humans evolved: by adapting our visual system to the statistics of nature [Barlow, 1961]. Having said this, one could make a distinction between datasets and tasks that could lead to more human-like behavior than others. For example, it is to be expected that a model trained to discriminate between handwritten digits may present fewer human-features than a model trained to detect persons in a video image, as the last one is more similar to what humans have been doing for a long time and has been ingrained into our brain through the years.

In [Kumar et al., 2022], they define a "Perceptual Score" and use it to evaluate whether a better performance in a classification task leads to a better Perceptual Score. Their work focuses on testing models of different kinds (EfficientNets, Vision Transformers, ResNets, and AlexNet) trained to perform classification on ImageNet, and what they find is that *better ImageNet classifiers achieve better Perceptual Scores up to a certain point. Beyond this point, improving on accuracy hurts Perceptual Score*, as can be seen in Figure 3.5. This could indicate that, maybe, a model becomes more human as it trains, but its "humanness" can go away in favor of improving its performance at a task by overfitting it.

[Hernández-Cámara et al., 2023] extends previous works to assess not only the performance of different architectures on a single task but to test how their Perceptual Score is affected by the training goal, the training data, and the read-out operation. According to our results, training with more than 1M natural images proved to be beneficial, which is on par with the idea that training with natural images should provide more human-like results. It's interesting to note that when comparing self-supervised and supervised tasks, supervised models presented higher Perceptual Scores.

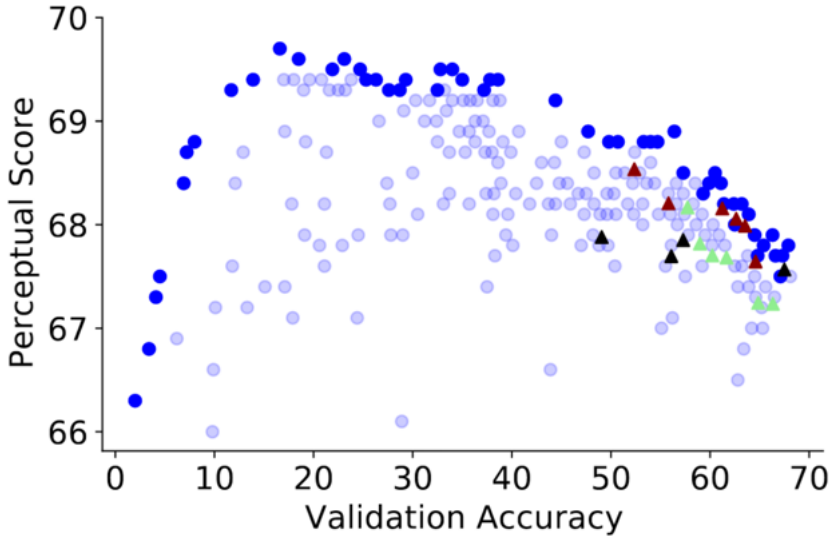


Figure 3.5: Perceptual Score obtained for different models that attain different accuracy scores on ImageNet classification. Surprisingly, an increase in validation accuracy is not directly correlated with an increase in Perceptual Score. Extracted from [Kumar et al., 2022].

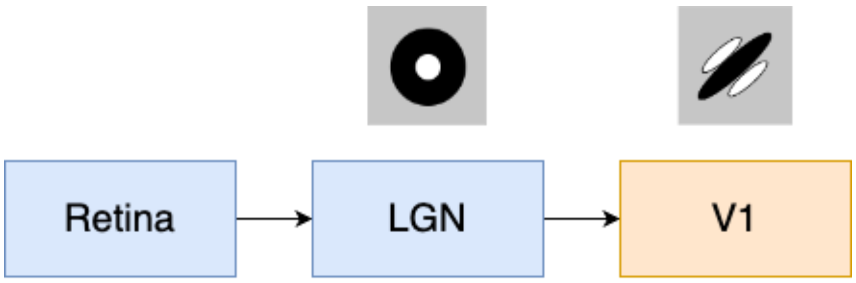


Figure 3.6: Schematic representation of the retina-LGN-V1 standard model of the visual system with their respective receptive fields (center-surround for retina-LGN and Gabor for V1. Receptive fields representations taken from [Carandini et al., 2005].

3.2.3 Models

The standard model of the early visual system is comprised of three stages: the retina, the LGN and the V1 cortex. It has been shown that the retina-LGN stage can be *satisfactorily described by the difference of two Gaussians* [Enroth-Cugell and Robson, 1966], while in the cortical region *the most efficient stationary stimulus was termed and edge* [Hubel and Wiesel, 1962], which corresponds to having Gabor-like receptive fields in V1. An schematic representation of the standard model is shown in Figure 3.6. An additional non-linear stage was then proposed in [Carandini and Heeger, 1994] consistent on dividing the activity of a single cell by a pooling of the ther single cells, which corresponds to a form of Divisive Normalization.

Following we will go over three models that were based on ideas taken from the human visual standard model: Normalized Laplacian Pyramid Decomposition or NLPD [Laparra et al., 2016], Linear+Nonlinear cascade [Martinez-Garcia et al., 2018] and PerceptNet [Hepburn et al., 2020].

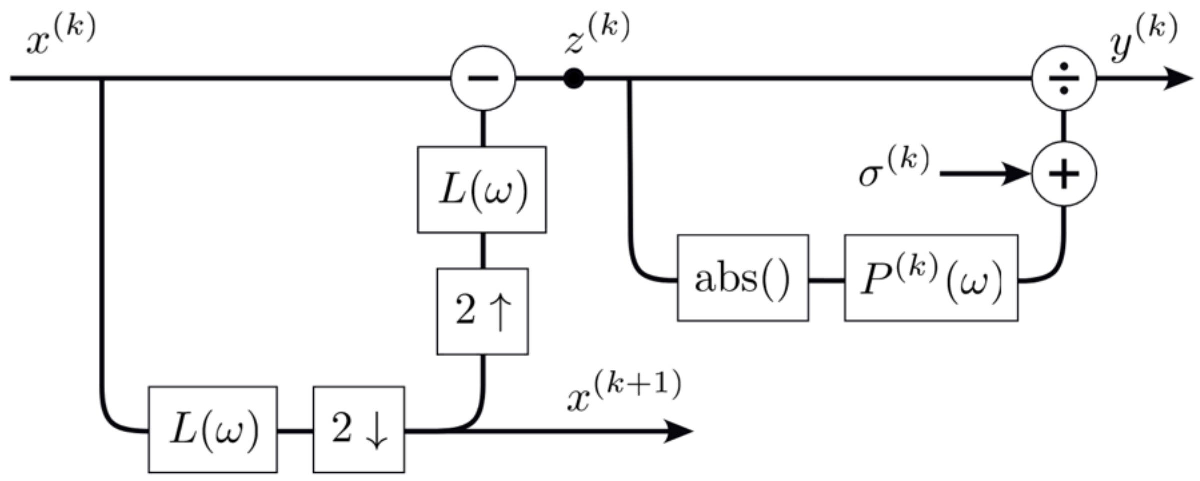


Figure 3.7: Schematic representation of NLPD for a single scale k. $L(w)$ + downsampling represents a Difference of Gaussians, while $P^{(k)}(w)$ corresponds to applying a Gaussian. x^{k+1} is used as the input for the next scale. Extracted from [Laparra et al., 2016].

Normalized Laplacian Pyramid Decomposition (NLPD) [Laparra et al., 2016]

In this regard, an *image quality metric based on the transformations associated with the early visual system* was proposed in [Laparra et al., 2016], where they take inspiration from the human visual path to construct a pyramid of transformations that apply local luminance subtraction and local gain control. This new image quality measurement is based on applying center surround filtering to the inputs, reminiscent of the properties seen in the retina and thalamus. They show that both stages *lead to a significant reduction in redundancy relative to the original image pixels* consistent with efficient coding [Barlow, 1961, Malo and Laparra, 2010], beating state-of-the-art models of that time. A schematic representation of a single scale Normalized Laplacian pyramid is shown in Figure 3.7.

A Linear+Nonlinear cascade [Martinez-Garcia et al., 2018]

In an effort to move towards a model of the visual system that offers useful analytical insight into the psychophysics, the physiology, and the function of the visual system, [Martinez-Garcia et al., 2018] proposes a *cascade of Linear+Nonlinear transforms*, which are known to be very successful in modeling a number of perceptual experiences. One of its advantages versus the other models is that, as it employs known functional forms inspired by the human visual system, its inverse can be calculated and studied, providing a lot of insight into the model and expanding greatly the analysis and explainability of the model. Something that deep learning-centered models fail to attain to this extent. The model contains a sequence of modules *that account for brightness, contrast, energy masking, and wavelet masking*, and is built around the idea of modularity so that different properties can be checked by plugging and removing modules independently. Its nonlinearities are based on canonical divisive normalization, which is also a biologically inspired function. The Linear+Nonlinear cascade of modules and their inputs and outputs is shown in Figure 3.8.

When applying this model to the TID2008 database they obtain state-of-the-art results ($\rho_p = 0.88$) but in [Martinez-Garcia et al., 2019] they argue that focussing on the

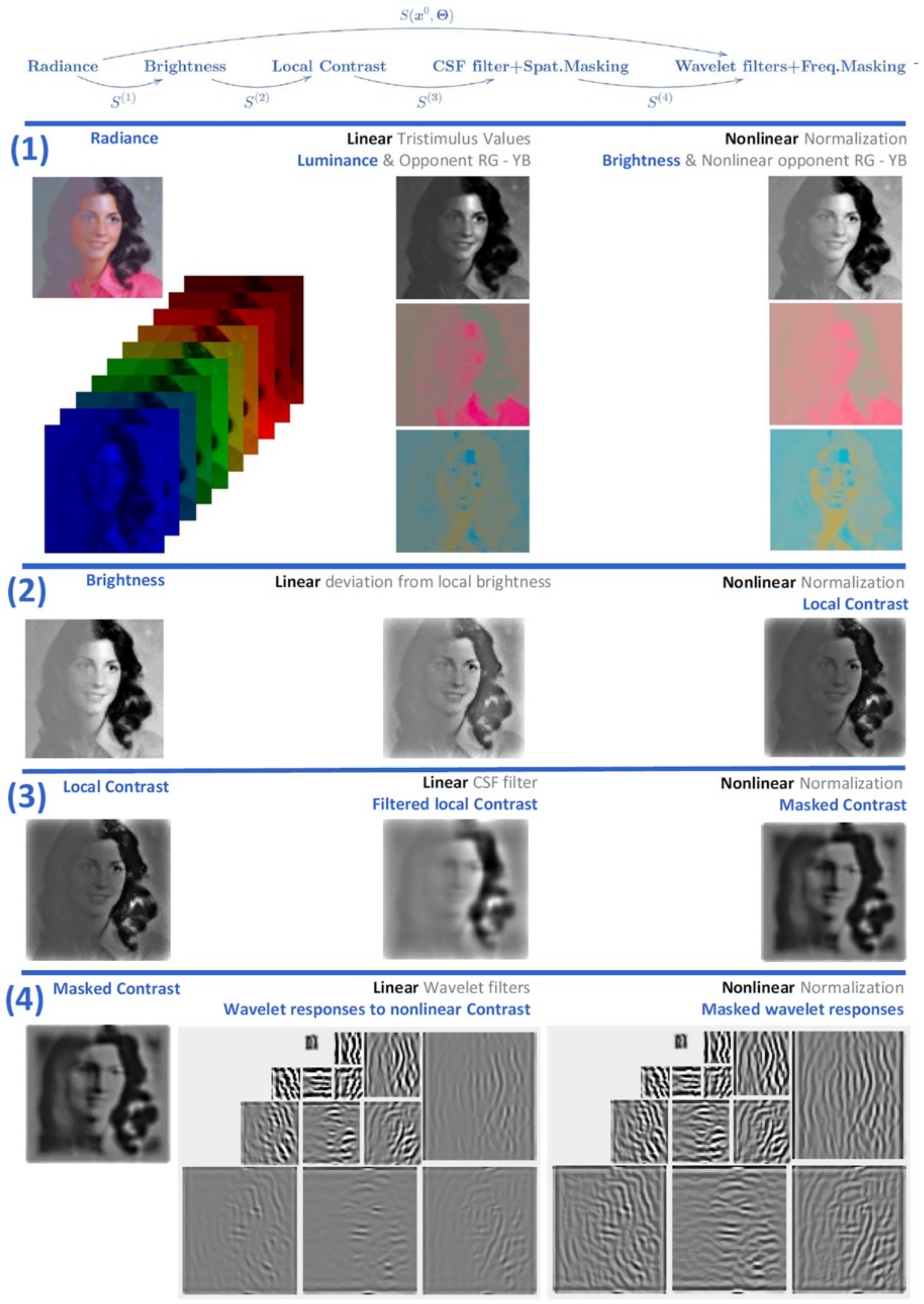


Figure 3.8: On the top, a simplified view of the cascade of Linear+Nonlinear modules. Below is an example of the inputs & outputs for the different modules when using a sample image as input to the model. Extracted from [Martinez-Garcia et al., 2018].

correlation only missed reproducing certain phenomena, so they added extra frequential constraints in the Gabor domain and in the kernel of the Divisive Normalization in order to reproduce certain psychophysical behaviors. The modularity of the model allows the authors to test the model with different selections of models, proving that each subsequent stage helps to improve the previous result thus, implying that adding human visual-inspired stages into the model is beneficial to its performance, at least in image quality assessment tasks.

PerceptNet [Hepburn et al., 2020]

While both previous methods included explicit functional forms (center-surround in [Laparra et al., 2016], CSFs, center-surround and Gabor filters in [Martinez-Garcia et al., 2018, Martinez-Garcia et al., 2019]), the most recent trend in image quality assessment has been moving into the usage of deep neural networks that were fitted on perceptual databases as per the usual deep learning "brute force" approach. Two of the methods that could be considered state of the art in the field are LPIPS [Zhang et al., 2018] and DISTS [Ding et al., 2020], and both of them are based on a VGG network with no specific biological nor human-like behavior in them. To leverage the phenomenal capabilities of deep neural networks but retain what made previous methods SOTA at their times, [Hepburn et al., 2020] introduces PerceptNet, *a convolutional neural network where the architecture has been chosen to reflect the structure and various stages in the human visual system.* The idea behind it is to avoid stacking layers while hoping for better performance, but to use operations that have a correspondence with a process happening in the human visual system. Adding to it, instead of using the well-known ReLU non-linearity that is so present in ANNs, it implements a divisive normalization nonlinearity, which is also inspired by the human visual system, and it's seen that its use can *increase the ability of the network to judge perceptual similarity.* Following Figure 3.9, the architecture is defined so that it could accommodate the following stages:

1. Gamma correction and opponent color space transformation.
2. Von Kries transform and center-surround filtering.
3. LGN normalization.
4. Orientation and multiscale sensitive transform (Gabor filters) and Divisive Normalization in V1.

Choosing an architecture so heavily based on the visual path system is a big step towards training biologically-inspired models but it may not be enough. In [Vila-Tomás et al., 2022a, Vila-Tomás et al., 2022b], we studied the psychophysical properties of PerceptNet and found that its learnt filters didn't converge to the expected transformations they were based on. For example, the receptive fields of the last convolutional layer of the network are shown in Figure 3.10, where it can be seen that the layer that was supposed to perform an orientation sensitive transform (i.e. Gabor filters) did get a set of filters that showcased a lot of repetitions and no particular structure. In this same work, we didn't obtain the psychophysical properties that were to be expected from a model that had the intented transformations stated before.

This served heavily to support our claim that choosing an achitecture and training all the convolutional filters freely wasn't enough to obtain models which had specific

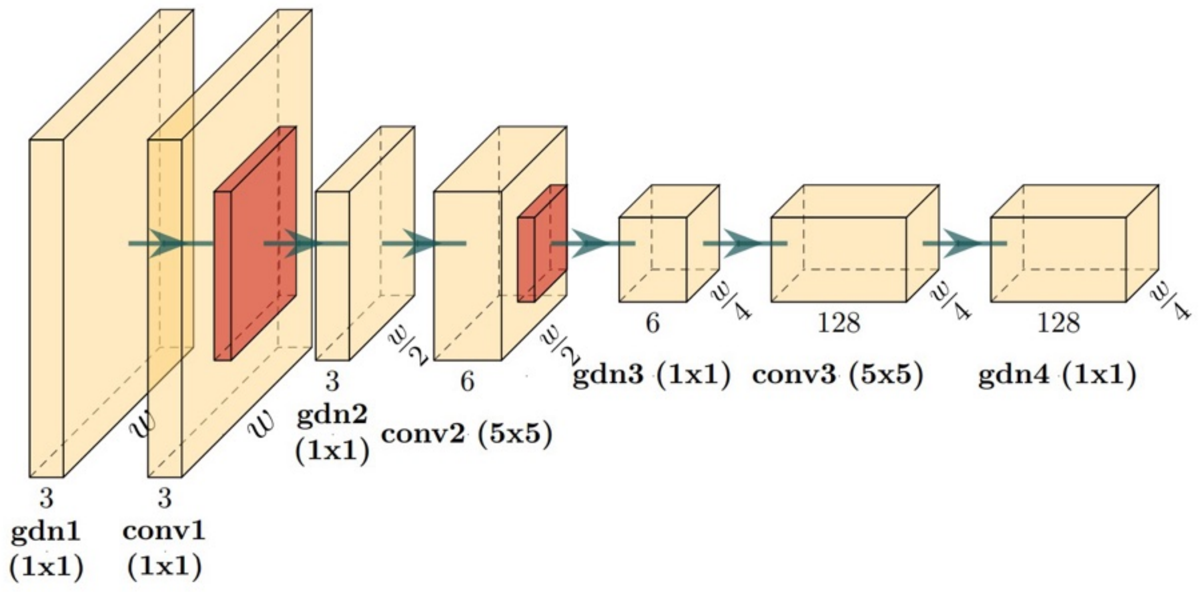


Figure 3.9: PerceptNet. Extracted from [Hepburn et al., 2020].

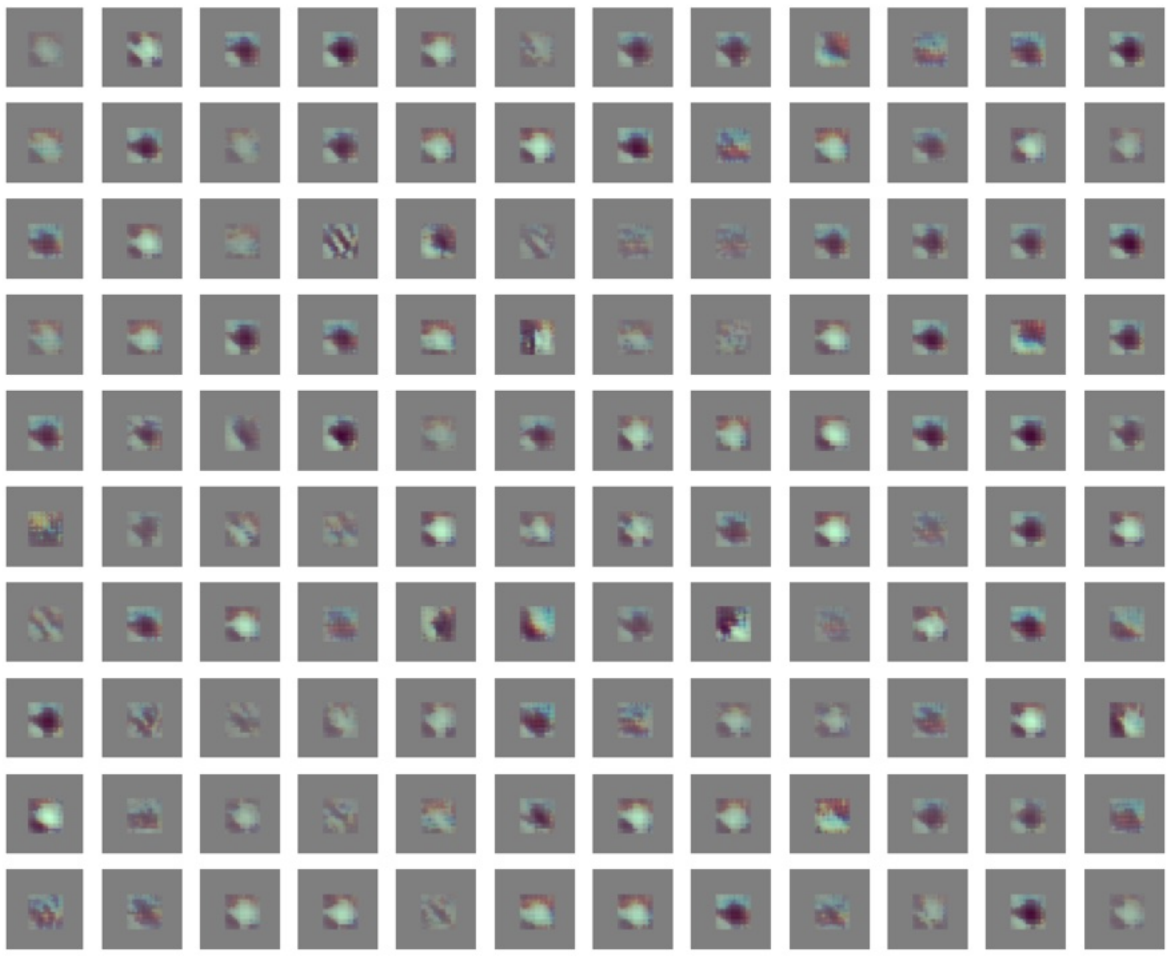


Figure 3.10: Receptive fields of the last convolutional layer of PerceptNet trained for image quality assessment. This layer was supposed to perform an orientation sensitive transform, but most of its learnt filters don't show those properties. Extracted from [Vila-Tomás et al., 2022a, Vila-Tomás et al., 2022b].

properties, but one could have more success if the desired functional transformations could be imposed into the model and optimized together through gradient descent. This may indicate that the set of filters that give the best performance for a given task are not the most human-like due to, for example, overfitting or particular biases of the data. What is expected nonetheless is that a fully biologically-based model that reproduces all the human psychophysics experiment should obtain a competitive performance in human perception tasks.

In this work we will take PerceptNet as a base model and impose certain parametric functions at specific layers in an effort to ensure that our model learns the desired set of transformations in order to reproduce properly human behavior. In Section 4.4.3 we dive deeper into the modifications made to the base model.

3.3 Bio-inspired functional forms

It is said that the ultimate test of whether the visual system is understood is predicting its response to arbitrary stimuli [Carandini et al., 2005]. In the early decades of visual neuroscience, the accepted standard model was the linear receptive field [Hubel and Wiesel, 1962]. This was based on a set of weights that were applied to an image in order to obtain an output through a weighted sum (what we called convolution in Section 3.1.2). A weighted sum is a simple approach that made the framework mathematically tractable and so it could benefit from the advances in image processing and linear system analysis. Within this approach, the output's magnitude of a visual system model would be higher as the input is more similar to its receptive field, while it would be very small if the inputs don't resemble it at all. Getting into more detail, the retina and LGN are known to have center-surround receptive fields whose objective is to *tune for the spatial frequency of a drifting grating* [Enroth-Cugell and Robson, 1966], while the filters in V1 are known to perform orientation tuning and present an elongation along one spatial axis [Hubel and Wiesel, 1962]. Both of these receptive fields can be modeled with a difference of Gaussians and a Gabor function respectively.

In Section 3.2.3, PerceptNet's original architecture was set in a way that resembled the retina-LGN-V1 path but all the weights of the linear transformations were left free. Instead, we could use the available biological information to constrain the weights so that they match what we know about the visual system. More about this in Section 4.4.3. Besides the benefits of having a linear approximation, *there are nonlinear phenomena that can't be explained by a linear receptive field alone* [Carandini et al., 2005]. This motivated the addition of a second stage to these models that transforms the receptive fields' outputs into firing rate responses, taking the form of a nonlinearity that depends only on its instantaneous input. Two notorious forms of nonlinearity are the rectification and the saturation, which conform to the three most common activation functions in artificial neural networks as we will see in Section 3.1.3. *The combination of a linear filter and a static nonlinearity creates the linear-nonlinear model of spiking neurons* [Carandini et al., 2005], which serves as inspiration for the computations in artificial neural networks and was as well the building block on which the models in Section 3.2.3 were built upon.

The following sections will go briefly through some works that have implemented some of these biological concepts into artificial neural networks.

3.3.1 Center Surround

The appearance of center-surround-like filters in computer vision dates before the bloom of deep learning. Edge detection is a crucial task when analyzing images with computer vision and the usual approach to building edge detection filters is using Differences of Gaussians (DoG) which incidentally corresponds to using a center-surround cell [Assirati et al., 2014]. It's worth remembering that center-surround cells were also used in [Laparra et al., 2016, Martinez-Garcia et al., 2018], as we already saw in Section 3.2.3. Following a bio-inspired approach, [Hasani et al., 2019] *designed new connections between units and their surroundings in CNNs to achieve more biologically plausible networks*. Introducing center-surround operations appears to improve the performance of the models in object recognition tasks at the same time improves the training speed of the model. Note that their implementation is fully hard-coded and doesn't add more parameters to the model. Building on this work, [Babaiee et al., 2021] *extends the receptive field of convolutional neural networks with two residual components* resembling the on-center and off-center pathways in the visual system. Their main addition is that they compute the hyperparameters of the DoG analytically from the size of the receptive fields. What they find is that introducing edge detection biases in their models improves their performance and makes them more robust to changes in lighting conditions.

3.3.2 Gabor

As happened with center-surround cells, the use of Gabor filters as feature and edge detectors have been widely spread in the computer vision community for a long time [Mehrotra et al., 1992]. It was even shown in [Bergstra et al., 2011] that one could *outperform standard sparse coding* by employing a dictionary with only a single element: a Gabor-like filter. When working with images, traditional sparse coding leads to overcomplete bases of localized edge detectors at many locations, scales, and orientations that require a lot of data to be trained but, by incorporating the Gabor prior into the problem, the sparse coding dictionaries can be learned from much fewer data because they have only one element.

Within the deep learning community, there have been several works [Calderón et al., 2003, Kwolek, 2005, Luan et al., 2018] that try to introduce Gabor filters into convolutional neural networks, but all of them resort to using them as fixed feature extractors in the first layers to reduce the number of parameters and make the networks more robust. [Alekseev and Bobe, 2019] is the first work that proposes including the Gabor parameters in the optimization process and optimizing its parameters jointly with the rest of the network with gradient descent. Based on previous approaches, they only use Gabors in the first layer of their network, leaving the rest of the layers free as in conventional convolutional neural networks. Additionally, [Pérez et al., 2020] explores *the effect on robustness against adversarial attacks when replacing the first layers with Gabor layers*, showing that they obtain a consistent boost in robustness over regular models. They go a step further and introduce a regularizer to be used during training to *further enhance network robustness*. A main difference with respect to [Alekseev and Bobe, 2019] is that they pre-select a set of rotations θ_j and avoid training this parameter. When all the Gabor filters are built, they are convolved with every input channel, producing $C_{in} \times n_G$ feature maps, where C_{in} corresponds to the number of channels of the input and n_G corresponds to the number of Gabor filters built. These feature maps are activated with a ReLU function and combined via a 1x1 convolution to obtain a pre-selected amount of feature maps, C_{out}, which one could argue aren't strictly Gabor features but a linear combination of them.

3.3.3 Divisive Normalization

Divisive Normalization (in image quality), first proposed in [Teo and Heeger, 1994], is a canonical nonlinear saturating operation that has been widely studied as a biologically plausible function of what happens in the visual path after the linear stage and up to the V1 cortex [Carandini et al., 2005, Martinez-Garcia et al., 2018]. Its main feature is that it can act as a contrast normalization between features [Martinez-Garcia et al., 2018], and is different from the usual activation functions seen in artificial neural networks because it *computes the ratio of an individual neuron's response with respect to the summed activity of other neurons in its neighborhood* [Veerabadran et al., 2021].

[Laparra et al., 2010] takes the Divisive Normalization function and employs it as a Perceptual Metric *optimized for a restricted set of complex distortions with simple stimuli in the LIVE database*, showing that it can reproduce low-level psychophysics. As we have seen before, deep learning researchers also tried to introduce Divisive Normalization into artificial neural networks and, surprisingly, one of the first works to do it was [Krizhevsky et al., 2012], where they experiment with a form of divisive normalization that took into account the values in nearby channels when normalizing pixels in i a certain channel c and obtained an increase in generalization. Later, in [Simonyan and Zisserman, 2015] they decided not to use it in their final model (VGG) in favor of a ReLU nonlinearity because Divisive Normalization increased the computation time and didn't improve their results. It's worth noting that they only tested it in a small version of the model, so it is unknown if it would have improved the performance of the final model or not. In contrast with this, [Veerabadran et al., 2021] *combines divisive normalization with linear lateral interactions*, showing a contrast invariance when applied in a self-supervised task that wasn't seen in a VGG16 pretrained on ImageNet [Miller et al., 2022]. A difference with respect to [Krizhevsky et al., 2012] is that they don't consider different channels in the divisive normalization calculation. Improving on the neighborhood consideration, [Cirincione et al., 2023] implements the neighborhood as a 2D Gaussian whose parameters are optimized with gradient descent as in a traditional deep learning approach. Their model *shows not only a better accuracy but an improved robustness to common image corruptions, particularly contrast and fog.* The corruption to fog was also studied by [Hernández-Cámara et al., 2022], where they show that by introducing a Divisive Normalization after every convolutional block in a U-Net trained for semantic segmentation they improve the performance and also make the model much more robust to fog. They show this by training the model with images without fog and testing on three different fog levels, where it can be seen that the model with Divisive Normalization can recover some of the features hidden by the fog (Figure 3.11) and obtains a notably better performance on out-of-the-training distribution data.

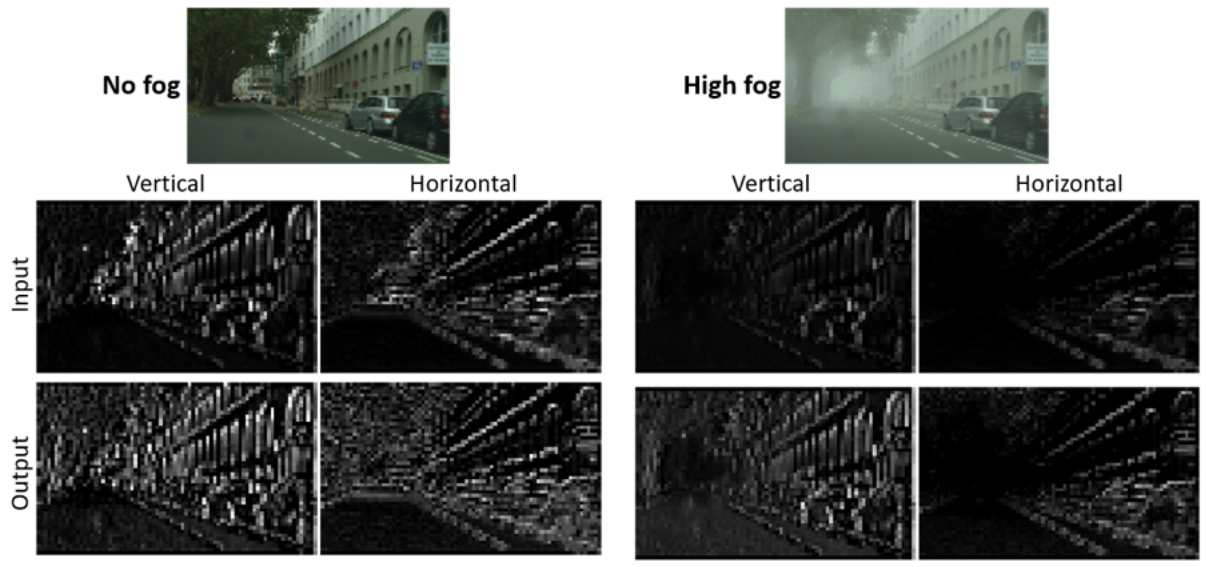


Figure 3.11: Effect of the second Divisive Normalization in a U-Net trained for semantic segmentation. It can be seen that, when corrupting the images with fog, the Divisive Normalization is able to recover some of the features that were present in the original image but were lost due to the fog. Extracted from [Hernández-Cámara et al., 2022].

Chapter 4

Materials and methods

In this Chapter, we will begin by mathematically introducing the three functional forms needed to reproduce the biologically-inspired receptive fields we have previously seen in Section 3.3 and are consistent with optimal coding results. This will give us the context for objective 1. Then, we will go through a set of simple toy problems that we can use to check our implementations and how these parametric forms behave when optimizing with gradient descent within a neural network, which constitutes objective 2. Finally, we will touch on the setup of the two main problems we want to solve: image classification and image quality, putting special emphasis on the image quality task. This consititutes objectives 3a and 3b, and their results will be discussed in Chapter 5.

4.1 Functional forms

After reviewing the field's landscape in Chapter 3 we know that we will need to implement an optimizable Gaussian in order to be able to build the center-surround cell as a difference of Gaussians and to consider (in a biologically plausible way) the neighborhoods in Divisive Normalization. Apart from this, we will also implement an optimizable Gabor to mimic the V1 receptive fields and the independent components of natural images. These functions can be defined for $\mathcal{R}^N$, but we will focus only on $\mathcal{R}^2$ because we will be working only with images.

Following the implementation of the most common deep learning frameworks, a convolutional layer that takes as input an image with C_{in} channels (i.e. $C_{in} = 3$ for RGB images) and outputs a single feature map, $C_{out} = 1$ it's said to have one filter F, but this filter is not a single bidimensional matrix. It is actually a tensor composed of C_{in} 2D matrices, and its output is calculated as $Output = \sum_i^{C_{in}} Input_i * F_i$. This is very important because, when building our functional layers, if we want to obtain C_{out} feature maps, we have to generate $C_{in} \times C_{out}$ functional kernels instead of only C_{out}. It is true that another option would be generating only C_{out} filters and repeating them to match the dimensionality of the input, which could induce some kind of regularization into the model but would reduce the generalization capabilities of these functions. We leave that for future exploration. A visual representation of this process is shown in Figure 4.1.

Another interesting fact relating these functional layers with conventional free-convolutions is that in free-convolutions, the weights can be of different magnitude for each corresponding input channel but when working with these parametric functions this is not so clear.

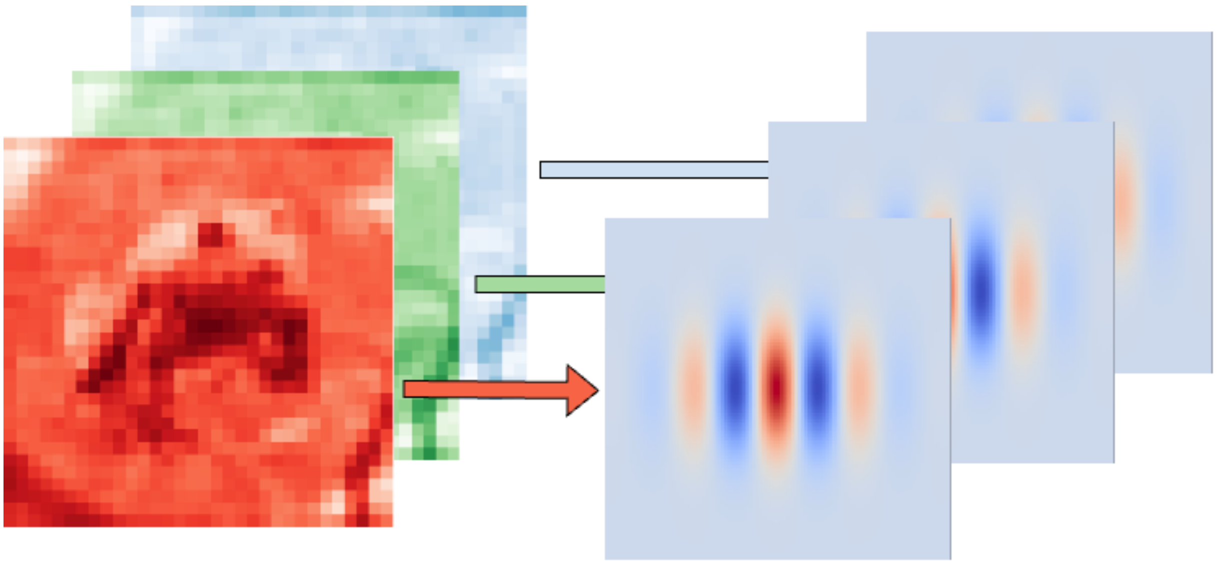


Figure 4.1: Sample image from CIFAR10 separated by its three RGB channels and three Gabor filters generated to be convolved with them. Showcases that we need to generate as many filters as input channels to generate a single output.

Two different filters generated by a parametric function will have different shapes but their magnitude won't be as different. In order to allow our model to weigh channels differently, we will add a parameter A that will multiply each generated filter: $\boxed{\hat{F}_i = A_i F_i}$, where F_i could be normalized (or not) in a particular way.

4.1.1 Gaussian

Gaussian functions have been broadly studied and are of a lot of interest in statistics and probability because a lot of phenomena can be characterized with Gaussian probability functions. As we're going to be working only with images, we will show the expression for the two-dimensional case in Eq. 4.1, where x_0 and y_0 represent the center point of the Gaussian (the location of its peak), and σ represents its width[1]. A visual representation of the function is shown in Figure 4.2.

$$G(x, y) = Be^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}} \tag{4.1}$$

When considering a Gaussian (or Normal) probability distribution, the coefficient B can be set to $B = \frac{1}{2\pi\sigma^2}$ so that the integral of the PDF is equal to 1. Its derivation, akin to normalizing the Gaussian so that it has a volume of $V = 1$, is shown in 4.2:

$$V = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G(x, y)dxdy = 2\pi B\sigma^2 = 1 \rightarrow \boxed{B = \frac{1}{2\pi\sigma^2}} \tag{4.2}$$

This normalization has more implications than only normalizing the probability since it will "pull down" wider Gaussians and "pull up" sharper ones, meaning that we could end up with very high or very low values in our filters if their width σ becomes too low or too

[1]We're considering only a symmetric Gaussian, but one could set different σ_i for each dimension so that it could take an ellipse form by considering a covariance matrix.

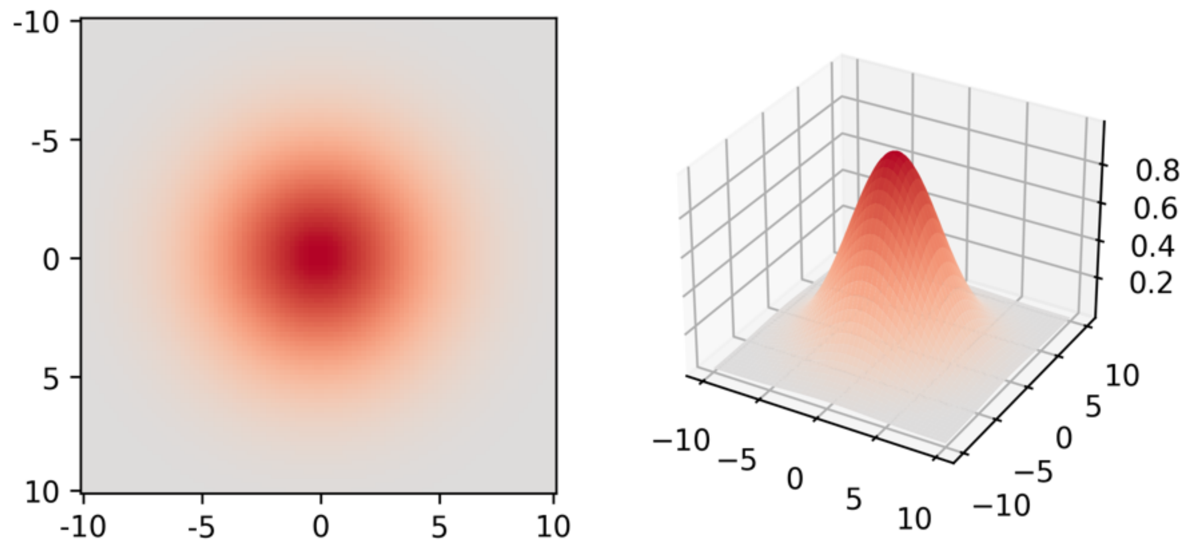


Figure 4.2: Gaussian function centered around $(0, 0)$ with $\sigma = 3.5$.

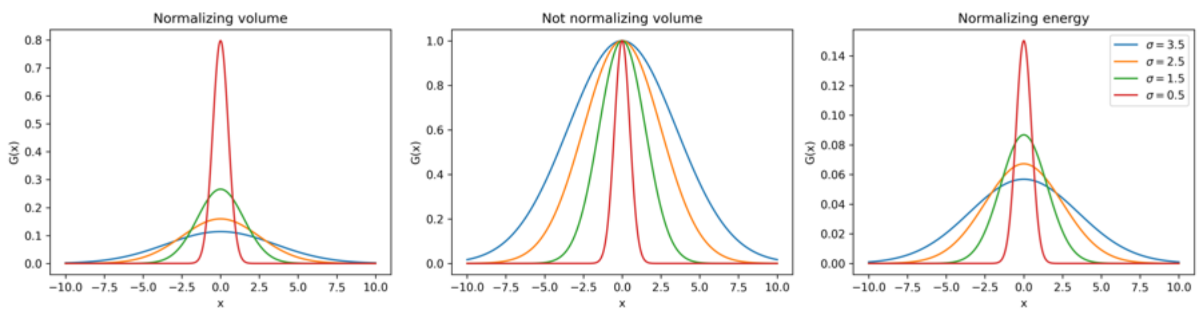


Figure 4.3: Comparison of 1D Gaussians when normalizing its volume (left), when not normalizing its volume (center), and when normalizing its energy. It's clear that normalizing the volume changes drastically the height of the Gaussian for different values of σ, while normalizing the energy has a similar effect but is less drastic. Normalizing the energy results in the lowest magnitude, which could be more aligned with the magnitude of common free-convolutional kernels.

high respectively. If we chose not to normalize the probability, the peak of the Gaussian would always have a value of 1 and only its width would be changed, while another option would be normalizing the Gaussian so that it has unit norm, which could be understood as normalizing its energy and would have similar effects as normalizing the volume but being less extreme. The three different cases are shown in Figure 4.3. The implications of this choice with respect to the training of the model will be analyzed in Section 4.2.1.

4.1.2 Center Surround (Difference of Gaussians)

The Center Surround receptive field may be defined by the Laplacian of a Gaussian (LoG), but it is often *approximated by a Difference of Gaussians (DoG) since it reduces the computational costs for two or more dimmensions* [Assirati et al., 2014]. Knowing this, it is trivial to define after having defined a single Gaussian, by taking the difference between two Gaussians of different widths and normalized probability:

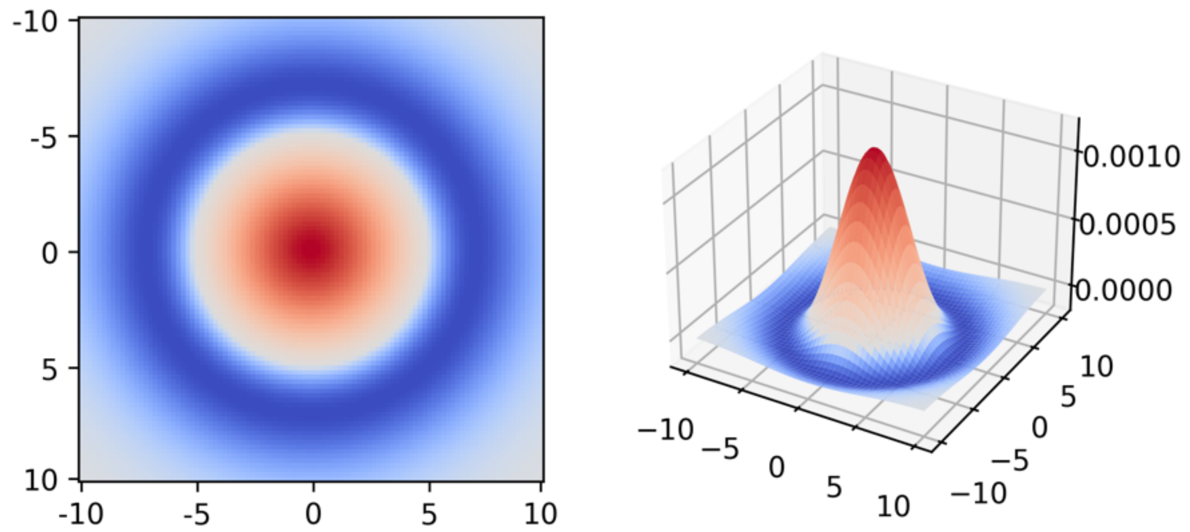


Figure 4.4: Center surround with a value of $k = 1.05$.

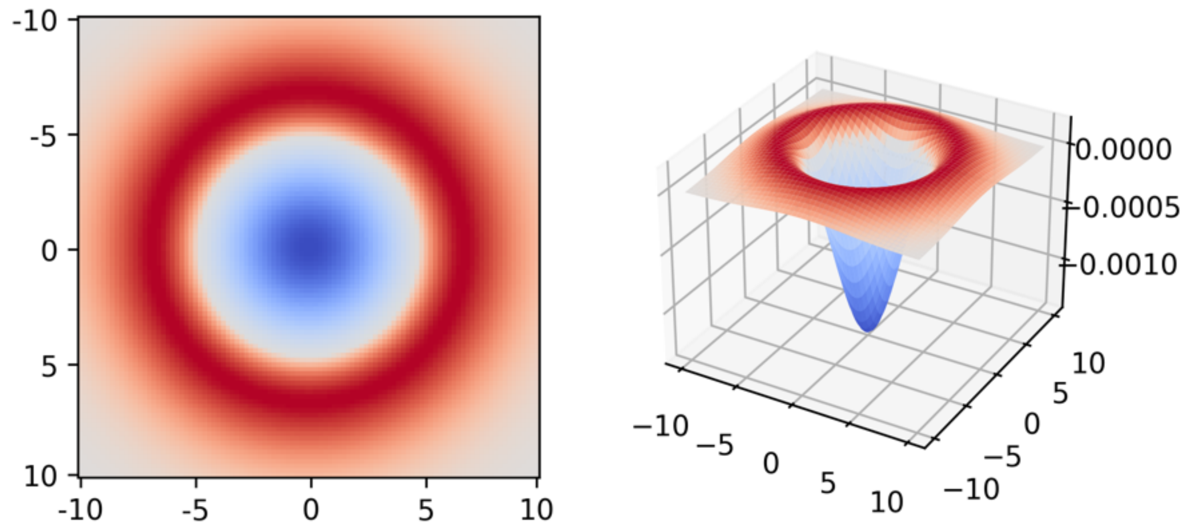


Figure 4.5: Center surround with a value of $k = 0.95$.

$$CS(x,y) = B_1 e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma_1^2}} - B_2 e^{-\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma_2^2}}$$

It can be handy to choose σ_1 and σ_2 so that $\sigma_2 = k\sigma_1$ while k is a number close to 1 [Evans et al., 2022]. When $k < 1$ we will have a downward center-surround (which could be interpreted as modelling an "off-centre" cell) and when $k > 1$, we will have an upward center-surround (which could be interpreted as modelling an "on" cell). We will discuss later the implications of this parametrization versus having two independent σ_i but, from an optimization point of view, the parametrized choice is easier to initialize in a meaningful way and can be better behaved when optimizing. Both cases can be seen in Figures 4.4 and 4.5.

Mention that you don't get the expected form if you don't normalize their individual probabilities before calculating the difference.

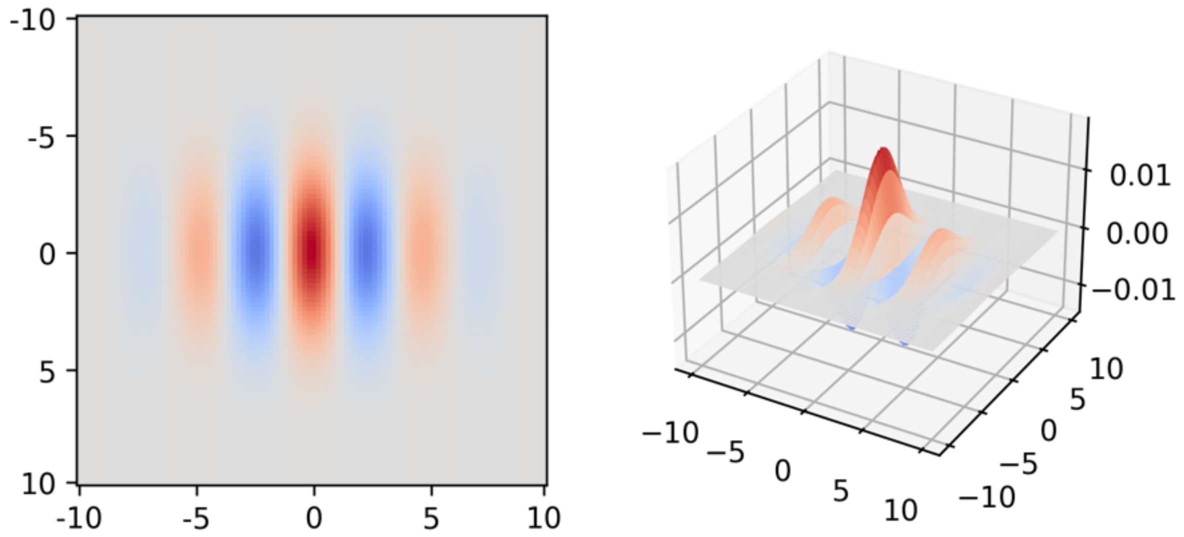


Figure 4.6: Gabor filter with $f = 20$ and $(\sigma_x, \sigma_y) = (3.5, 2.5)$.

4.1.3 Gabor

Gabor functions [Gabor, 1946] emerge from the necessity of having sinusoids located in space and while they were first defined in 1D, they were quickly extended to 2D in order to process images and extract features from them [Granlund, 1978]. They are built by composing a sinusoid with a Gaussian envelope like in Eq. 4.3, where $\vec{x'}$ corresponds to the domain $\vec{x}$ rotated by an angle θ ($\vec{x'} = R(\theta)\vec{x}$), Σ represents the covariance matrix, α indicates the angle of rotation of the Gaussian envelope and f the frequency of the sinusoid. Note that we will not consider the phase of the sinusoid because we will focus on V1 simple cells [Carandini et al., 2005], but incorporating the phase and building V1 complex cells would be a very promising extension in the future.

$$\mathcal{G}(\vec{x'}) = Be^{-\vec{x'}^T R(\alpha)^T \Sigma^{-1} R(\alpha) \vec{x'}} \cos\left(2\pi f \vec{x'}\right) \tag{4.3}$$

These kinds of functions are particularly useful because they have known frequency and orientation, allowing for interpretability of the models in a way that is not possible with usual convolutional neural networks, where the frequency and orientations of the learnt filters can be very hard to estimate. Its properties even allow building models that process differently different frequencies or orientations resembling a wavelet-like approach [Graps, 1995], something that isn't possible with non-Gabor filters either. An example of a Gabor functions with different parameters are depicted in Figures 4.6, 4.7, 4.8. It can be seen that the peak of the function is always at (x_0, y_0). This is due to the fact that we are using a cosine without phase. Expanding our work to allow changes in phase would generate slightly different filters that would make for a more complete base.

4.1.4 Optimizing for logσ

Something to keep in mind is that, when considering conventional convolutional filters, the values of the weights are not bounded, but that is not the case for all the parameters of the parametric functions we have shown before. Specifically, the parameter σ is positive definite because we can't have a negative width (it wouldn't make sense). In that regard,

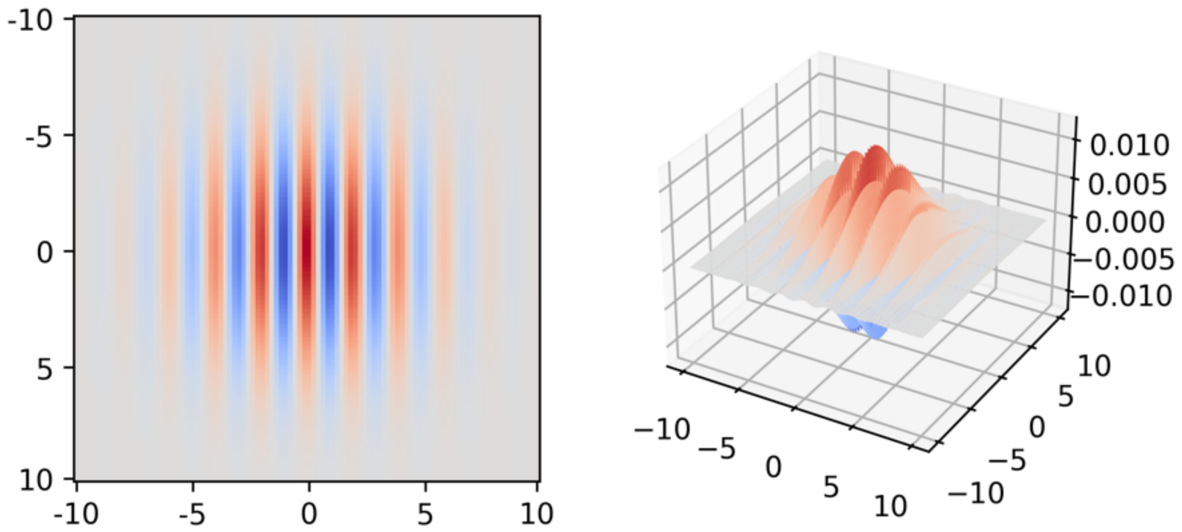


Figure 4.7: Gabor filter with $f = 50$ and $(\sigma_x, \sigma_y) = (3.5, 3.5)$.

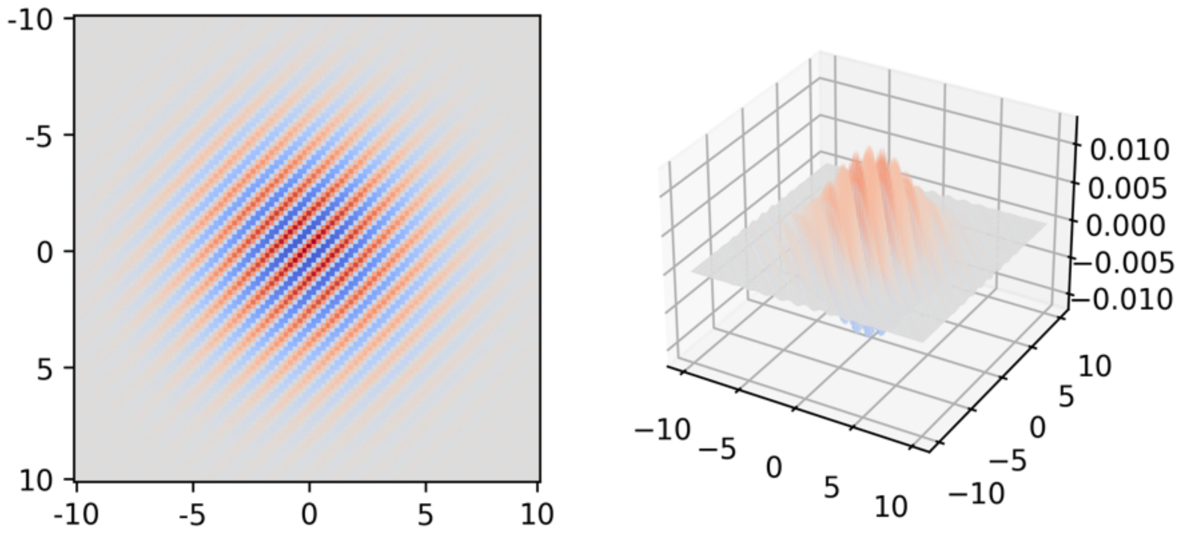


Figure 4.8: Gabor filter with $f = 20$, $(\sigma_x, \sigma_y) = (3.5, 3.5)$ and $\theta = 45°$.

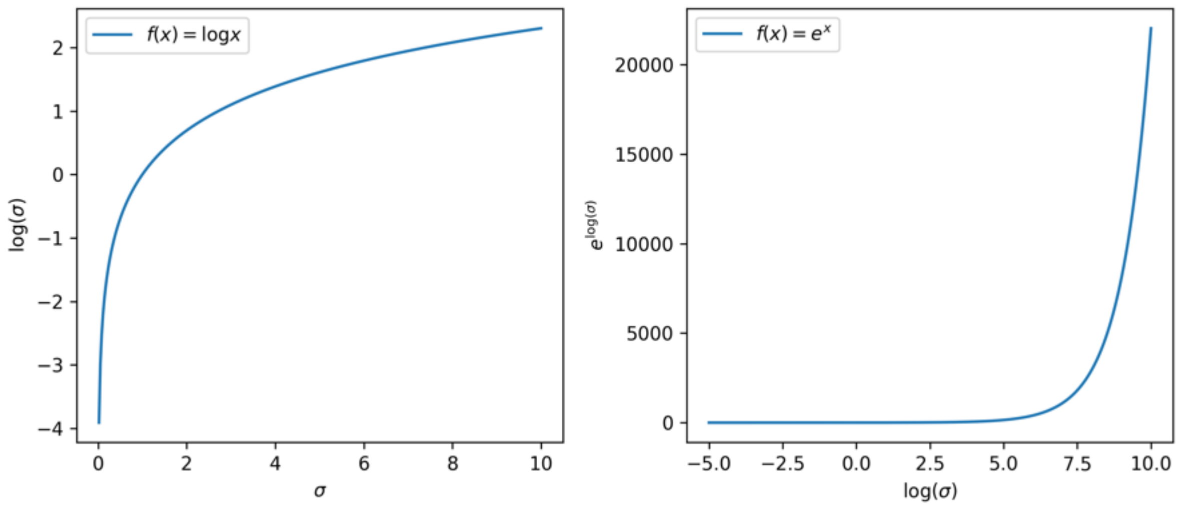


Figure 4.9: On the left: logarithmic function. It can be seen that its range is defined $x \in \mathcal{R}$. On the right: exponential function. Note that its domain is defined as $x \in \mathcal{R}$ but its range is defined as $x \in (0, \infty)$, recovering exactly the bounds we were looking for in the case of σ.

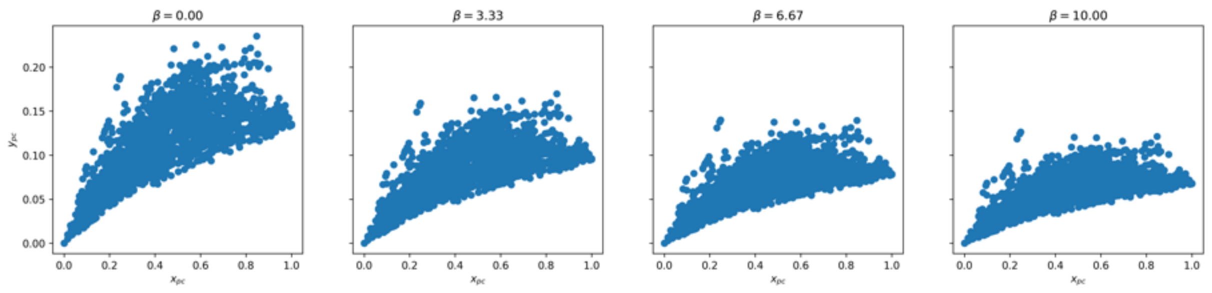
we could just clip it to a minimum value, but it doesn't perform well during training due to unstabilities when its value approaches 0, mainly due to the coefficient $B = \frac{1}{2\pi\sigma^2}$, and because the Gaussian (or the envelope in the Gabor filters) became of zero width, NaN values arised during training. A useful approach to solve this issue is to optimize for $\log\sigma$ instead, a trick that is also used when implementing Variational AutoEncoders [Kingma and Welling, 2013]. The usefulness of this transformation can be seen when considering that the logarithm transforms the optimization range from $\sigma \in [0, \infty)$ into $\log\sigma \in (-\infty, \infty)$, thus turning it into an unbounded optimization problem. It's also worth noting that this transformation also ensures that we don't get $\sigma = 0$, because when recovering the value of $\sigma = e^{\log\sigma}$, the exponential function is positive definite with an asymptote in 0, as shown in Figure 4.9.

4.1.5 Divisive Normalization

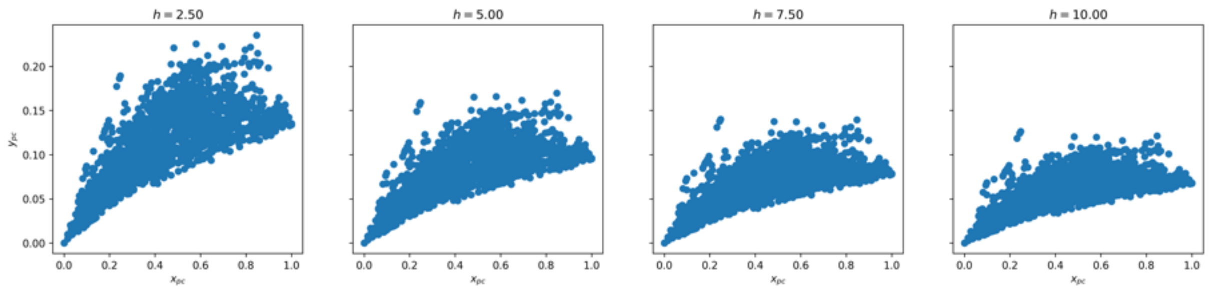
Divise Normalization is considered to be one of the neural canonical computations happening in the visual system. Consist of computing *a ratio between the responses of an individual neuron and the summed activity of a pool of neurons* [Carandini and Heeger, 2012], and can be expressed mathematically by Eq. 4.4, where α and ϵ are hyperparameters fixed at $\alpha = 2$ and $\epsilon = 1/2$, and β and H are a bias and convolutional kernel respectively:

$$y_{pc} = \frac{x_{pc}}{\left(\beta_c + H_{p-p'cc'} x^{\alpha}_{p'c'}\right)^{\epsilon}} \tag{4.4}$$

Although it may seem daunting, this computation can be implemented very easily with a usual convolution of kernel H and bias β by calculating the convolution over the input x, which corresponds to performing a weighted sum of the neighborhood for each pixel in the input. The exponents α and ϵ are usually fixed because they are not well-behaved when optimizing with gradient descent, but the bias and the kernel can be left free or constrained to one of the functional forms that we have already seen. A common approach



(a) Bias



(b) Kernel

Figure 4.10: Pixels of a sample image before and after applying Divisive Normalization (x and y axes respectively). When changing the value of the Bias (β) the kernel's magnitude is set to $H = 1$, and the value of the Bias is set to $\beta = 1$ when changing the magnitude of the kernel H.

[Watson and Solomon, 1997] and what we will do is constraining it to a Gaussian kernel, enforcing that closer pixels will affect more than those afar, while the different channels interact densely[2] as stated in Eq. 4.4.

An interesting property of this function is that we can control "how" non-linear it is with the magnitude of β and H (α and ϵ also affect this, but we will not explore them because they are going to be fixed). Figure 4.10 presents the results of applying the Divisive Normalization to a natural image (the same frog image from CIFAR10 as in Figure 4.1) with different values of β (a) and H (b). Note that H was set as a (3×3) kernel of 1s multiplied by a factor h: $H_{ij} = h$. In this Figure we can see that the higher the magnitude of β or H, the bigger the denominator will be, and the lower the output will be.

One problem with the formulation in Eq. 4.4 is that, when the denominator gets big, the point distribution gets squeezed to lower values (see rightmost scatter plots on Figure 4.10). In order to tackle this, [Martinez-Garcia et al., 2019] introduces a multiplicative constant leading to Eq. 4.5:

$$y_{pc} = \left(\beta_c + H_{p-p'cc'} {x^*_{p'c'}}^{\alpha}\right) \frac{x_{pc}}{\left(\beta_c + H_{p-p'cc'} x^{\alpha}_{p'c'}\right)^{\epsilon}} \tag{4.5}$$

This modification ensures that $y_{pc} = x^*_{pc}$ for a given x^*_{pc}, alleviating the squeeze effect and improving the performance. The inclusion of this modification is not studied in this work but could be of great interest in subsequent projects.

[2]This means that every channel can interact with every other channel without any restrictions or penalties.

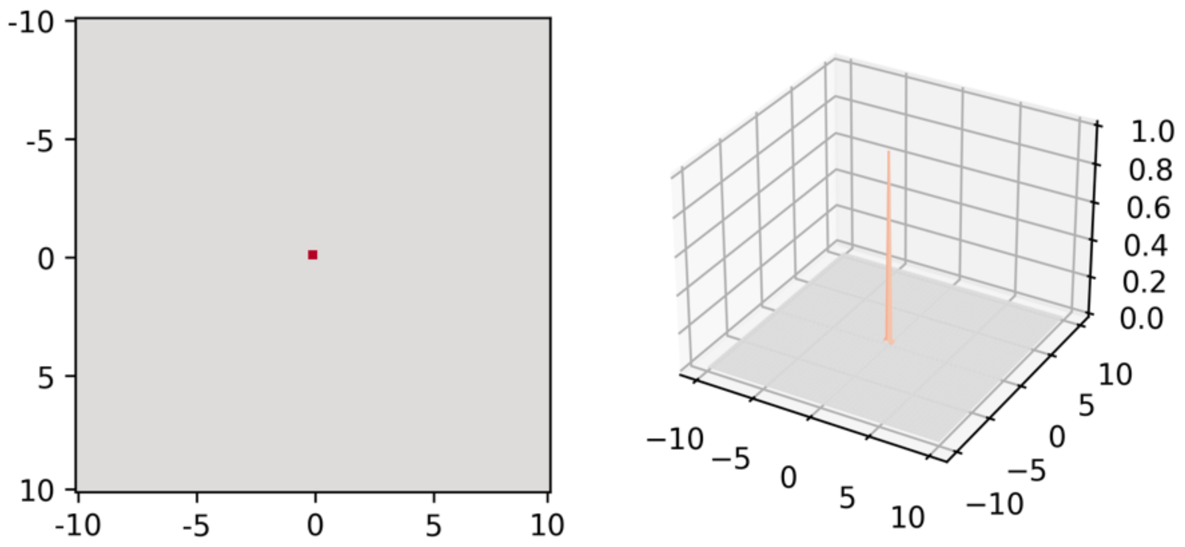


Figure 4.11: The solution for a reconstruction task when applying convolutions is a 2D Delta function. It can be approximated by a Gaussian where $\sigma \to 0$.

4.2 Toy examples

The approach we're taking is slightly deviating from the usual deep learning approach (where every pixel of the convolution kernels is trained separately) which has been broadly studied and optimized to work almost flawlessly out of the box, so it is expected that we have to consider additional factors when dealing with functional layers. To be able to assess these deviations and check that our implementations are as bug-free as possible, we have come up with a set of toy problems that have known solutions so that we can make sure they are operating as desired. To this effect, we have designed two toy tasks: (a) reconstruction using a single Gaussian and (b) rotation prediction using four Gabor filters.

4.2.1 Reconstruction

The problem

The simplest problem that can be solved with a Gaussian and has a known solution is the reconstruction task. Given an input vector $\vec{x} \in \mathcal{R}^{n,n}$ representing an image with only one channel, a reconstruction problem is one such that a function f applied to $\vec{x}$ minimizes the loss $\mathcal{L} = ||f(\vec{x}) - \vec{x}||_2$.

Known solution

In general, the solution to this problem is the identity function, $f(\vec{x}) = \vec{x}$, which corresponds to a 2D Delta function (Figure 4.11) when constrained so that f has to apply a convolution of a kernel $\vec{k} \in \mathcal{R}^{s,s}$ over $\vec{x}$.

In the 2D case, its expression can be laid out as in Eq. 4.6.

$$\delta_{x_0,y_0}(x,y) = \begin{cases} 0, & \forall(x,y) \neq (x_0,y_0) \\ 1, & (x,y) = (x_0,y_0) \end{cases} \tag{4.6}$$

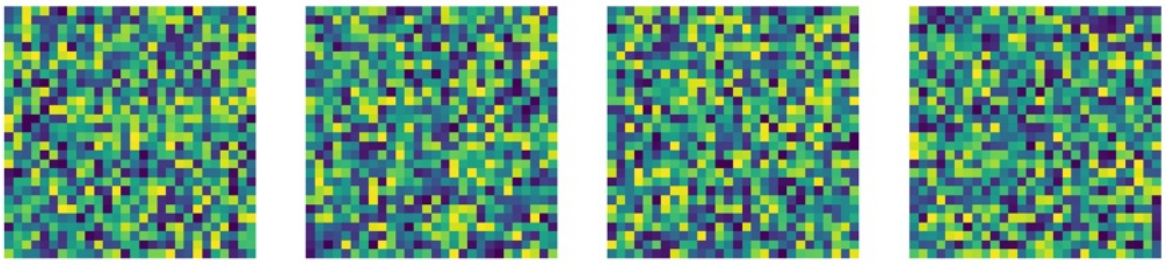


Figure 4.12: Sample inputs used in the reconstruction task. Each pixel is sampled from a uniform distribution $\mathcal{U}(0, 1)$.

We can show this by writing out the discrete convolution between $\vec{x}$ and δ_{x_0,y_0} in Eq. 4.7 and taking into account that δ_{x_0,y_0} is 0 everywhere but on $(i = x_0, j = y_0)$:

$$(\vec{x} * \delta_{x_0,y_0})[n,m] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \vec{x}[n,m]\delta_{x_0,y_0}[n-i, m-j] = \vec{x}[n,m] \tag{4.7}$$

To show that we can obtain this solution when imposing a Gaussian form, we can see that a Delta function is no more than a Gaussian function in the limit where its width goes to 0 (Eq. 4.8), so this solution is available in our constrained solution space.

$$\delta(x) = \lim_{\sigma \to 0} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-x_0)^2}{2\sigma^2}} \tag{4.8}$$

Keep in mind that the coefficient B tends to infinity, $B \to \infty$, when $\sigma \to 0$ so choosing an appropriate normalization may prove to be crucial to ensure a good optimization.

Obtaining the solution

Having set everything up, we only need a dataset to reconstruct. Actually, the expression in Eq. 4.7 doesn't depend on the particular $\vec{x}$ chosen, so we can use uniformly distributed noise as inputs to our model. Some examples of the input data are shown in Figure 4.12. As simple as this task may seem, its results can provide us with some interesting insights into the training dynamics of the layers we are introducing in this work.

The first interesting point we came across is that, when training for reconstruction with a single Gaussian layer (normalizing its volume), our model can quickly find a shortcut to minimize the loss: making the coefficient $A \to 0$. This can be seen in Figure 4.13. Considering that the optimal solution is $\sigma \to 0$ but this implies $B \to \infty$, it is to be expected that $A \to 0$ in order to compensate it. The problem in this case is that σ is not getting smaller but bigger, so we can assume that the model is finding a local minima (a shortcut) instead of behaving as intended.

Fixing the parameter $A = 1$ should be an easy way of blocking this shortcut but, funnily enough, it came up with a different one: making the Gaussian as wide as possible. Recalling Eq. 4.2, we see that, when normalizing the volume of our Gaussian, making $\sigma \to \infty$ would make $G(x, y) \to 0 \ \forall(x, y)$ and thus achieving the same results as making $A \to 0$. Figure 4.14 shows the training dynamics described above. Note that if we fix $A = 1$, the optimal solution is outside the solution space because the optimal solution when normalizing the volume should be $\sigma = 0$ and $A = 2\pi\sigma^2$.

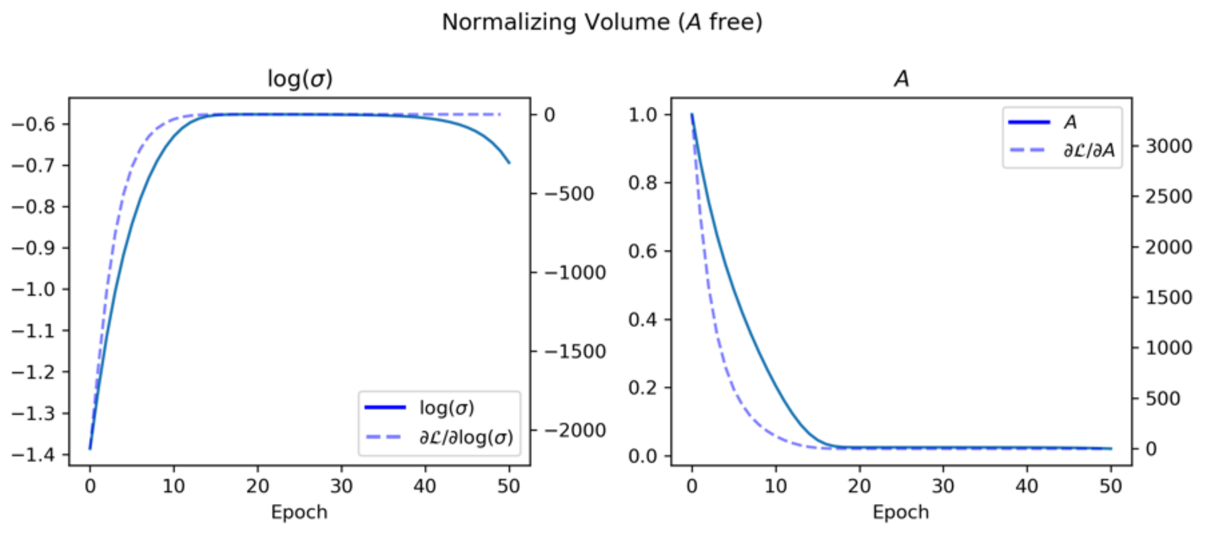


Figure 4.13: Evolution of $\log(\sigma)$ and A for a Gaussian reconstructing noise when its volume is normalized to $V = 1$. The left axis corresponds to the value of the parameter and the right axis corresponds to its gradient.

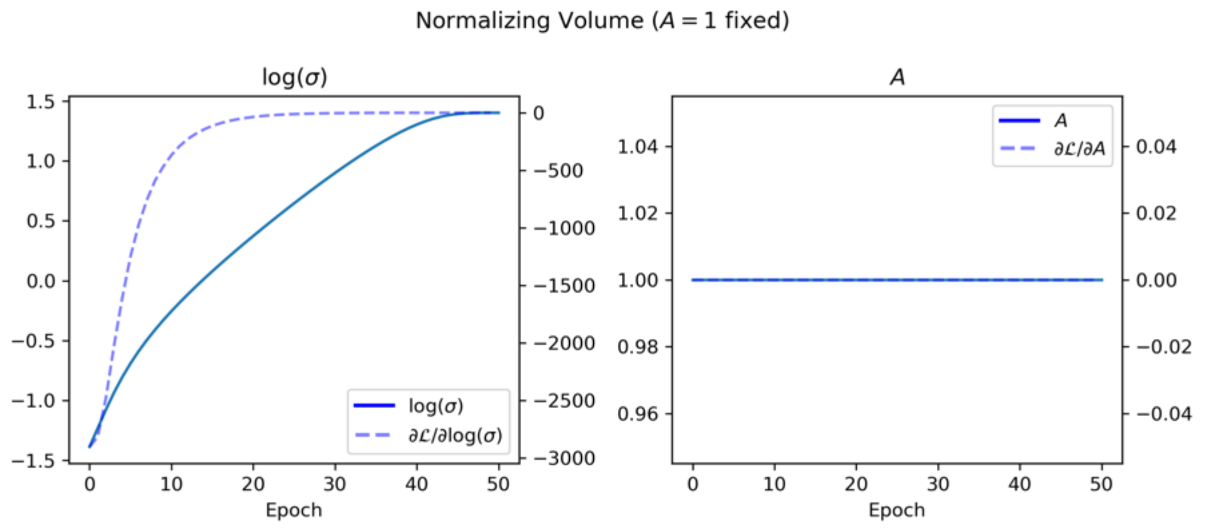


Figure 4.14: Evolution of $\log(\sigma)$ and A for a Gaussian reconstructing noise when $A = 1$ is fixed and its volume is normalized to $V = 1$. The left axis corresponds to the value of the parameter and the right axis corresponds to its gradient.

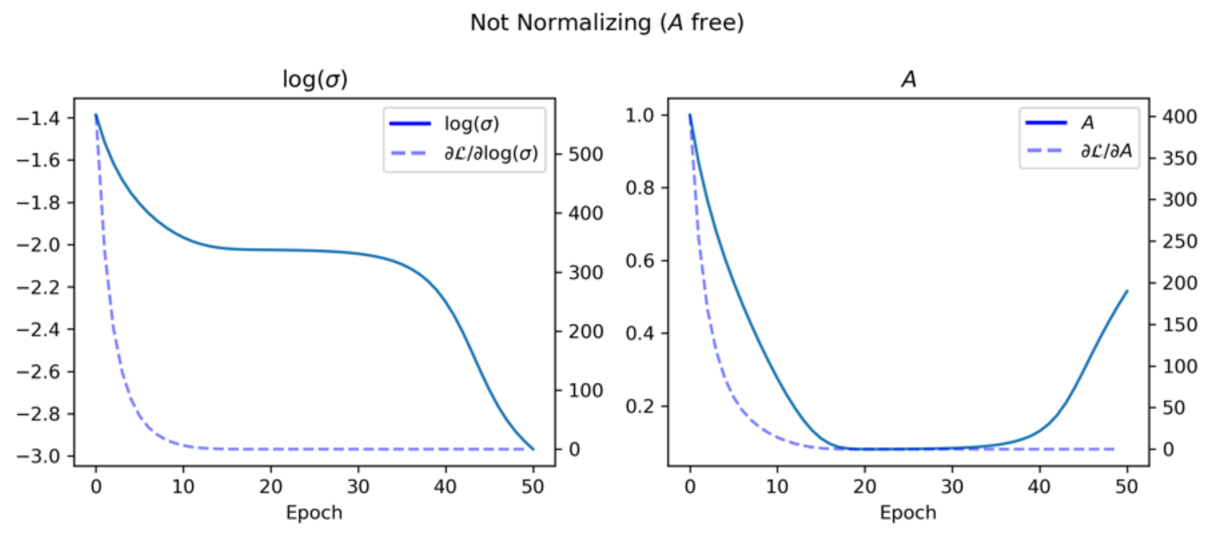


Figure 4.15: Evolution of $\log(\sigma)$ and A for a Gaussian reconstructing noise when no normalization is applied to the Gaussian filter. The left axis corresponds to the value of the parameter and the right axis corresponds to its gradient.

This result leads to experimenting with not normalizing the Gaussian to unit volume. This should help disentangle its relative height and its width, so the training should be better behaved. Doing so results in Figure 4.15, where a narrowing of the Gaussian is taking place (as we expected to happen since the beginning), but we also see a lowering in the factor A, indicating that a shortcut could be still being used. More interestingly, we see that at the end of the training, A has started to get bigger again.

Following our previous result, we repeated the experiment with a fixed $A = 1$, leading to a steady narrowing of the Gaussian (we don't see a plateau like we saw when A was free) and obtaining an even lower value of σ, as can be seen in Figure 4.16. We finally got the result we were looking for!

To wrap up this experiment, we realized that the gradient values might be a little too high compared with the usual values that are found in usual convolutional neural networks. This is related to the range of the activations of our parametric layers which are, indeed, orders of magnitude higher than those found in usual 2D convolutional layers. As it's known [LeCun et al., 2012], deep neural networks train best when the inputs and outputs of each layer are between the range $[-1, 1]$, so it would be very beneficial to control our layers in a way that could make their activations closer to that range. A first approach to this subject was normalizing the kernels so that their energy equals one, $E_k = ||k||_2 = 1$, where k represents a convolution kernel. With this final change, we repeat the previous experiments (Figures 4.17 and 4.18) and find that normalizing the energy of the kernels still leads to the desired results (even lower value of σ) but with more controlled gradients, which is exactly what we were looking for when applying this change. An evolution of the Gaussian filter during training is shown in Figure 4.19. See that, in opposition to the volume normalization, normalizing the energy of the Gaussian doesn't impose that $A \to 0$ when $\sigma \to 0$, allowing us to weight each filter properly.

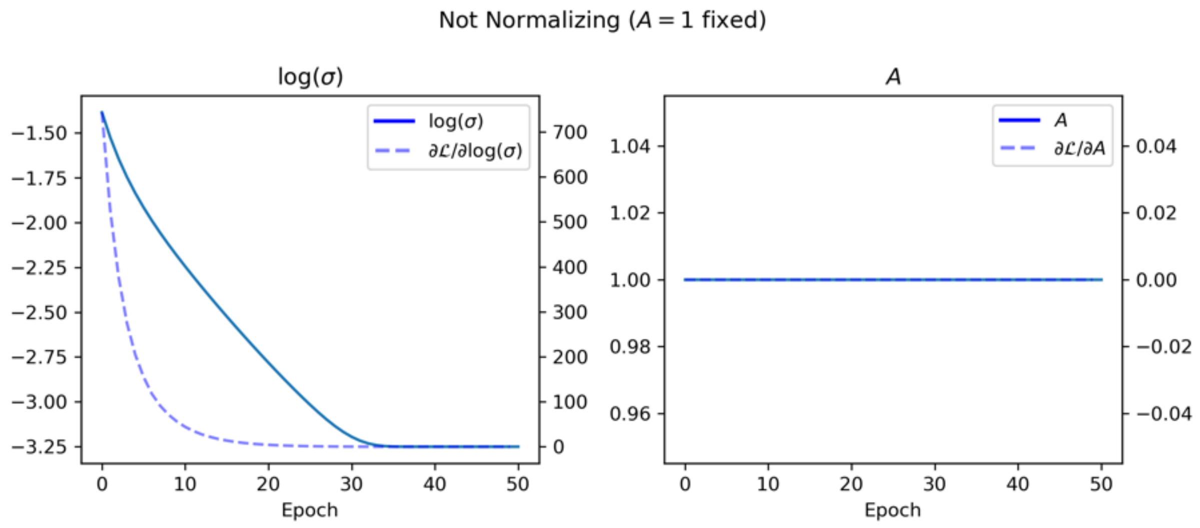


Figure 4.16: Evolution of $\log(\sigma)$ and A for a Gaussian reconstructing noise when $A = 1$ is fixed and no normalization is applied to the Gaussian filter. The left axis corresponds to the value of the parameter and the right axis corresponds to its gradient.

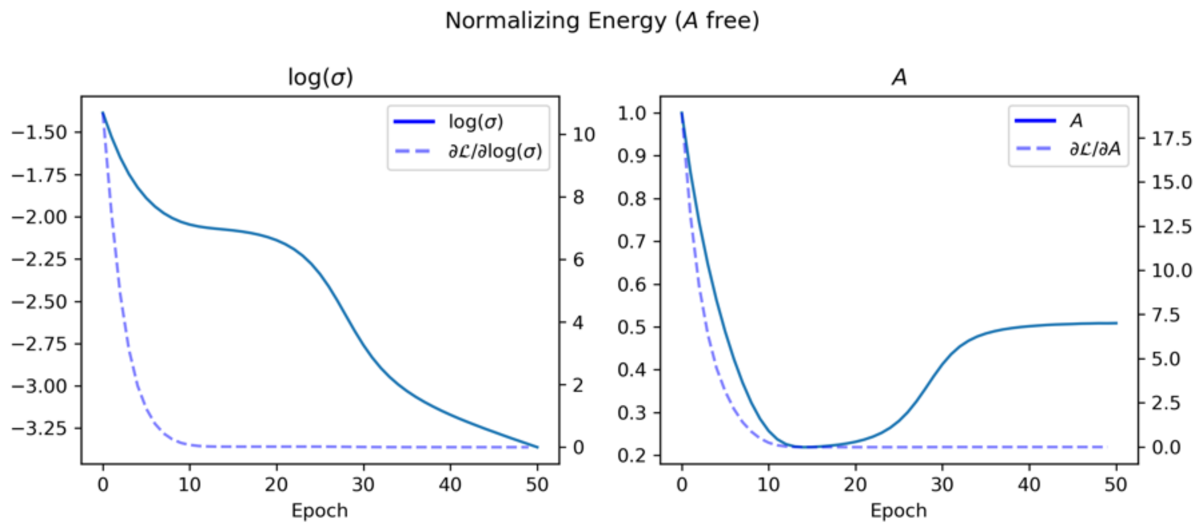


Figure 4.17: Evolution of $\log(\sigma)$ and A for a Gaussian reconstructing noise when its energy is normalized to $E = 1$. The left axis corresponds to the value of the parameter and the right axis corresponds to its gradient.

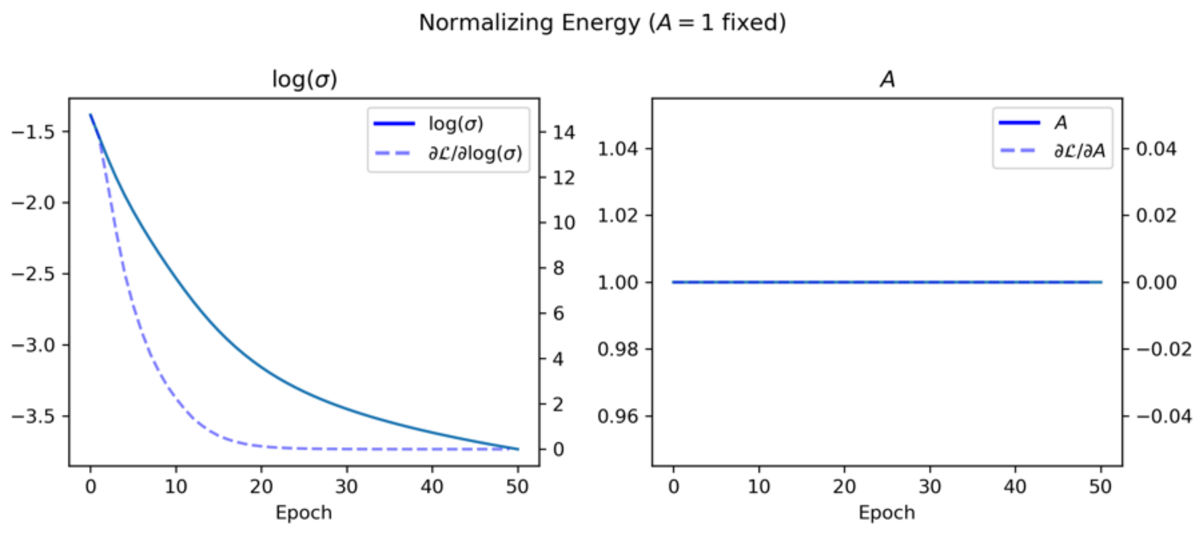


Figure 4.18: Evolution of $\log(\sigma)$ and A for a Gaussian reconstructing noise when $A = 1$ is fixed and its energy is normalized to $E = 1$. The left axis corresponds to the value of the parameter and the right axis corresponds to its gradient.

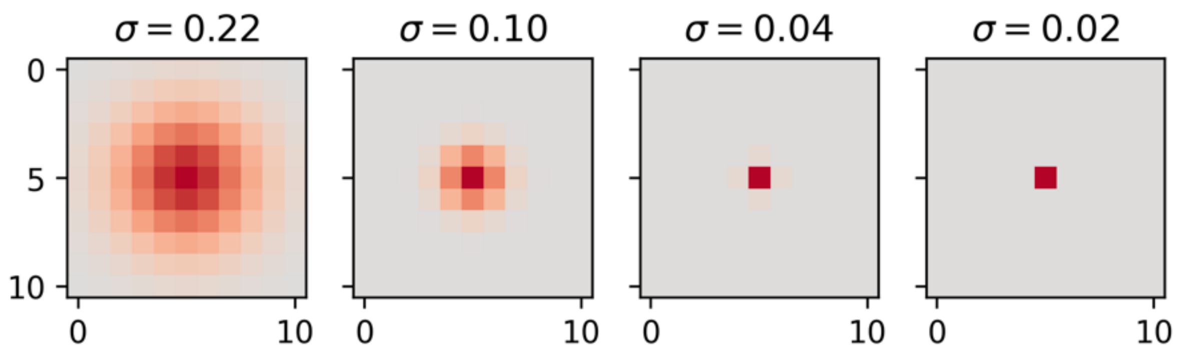


Figure 4.19: Evolution of a Gaussian filter whose parameter σ is optimized to perform reconstruction. We see the expected evolution of the Gaussian turning into a Delta. The values of σ are shown in degrees for a Gaussian filter comprising 1 deg.

Consequences

In the beginning, this experiment gave the impression that it was a very easy task that couldn't have more uses than testing our implementation but, in the end, it provided us with a lot of insight and information about the training behavior of these parametric layers that we can apply when moving forward into more complicated models. As a takeaway, we were able to solve this task obtaining good results and found that using a proper normalization of the filters is crucial. As an extra point, using a separate factor A to weight the filters may be convenient and is better behaved when normalizing the filters to have unit energy.

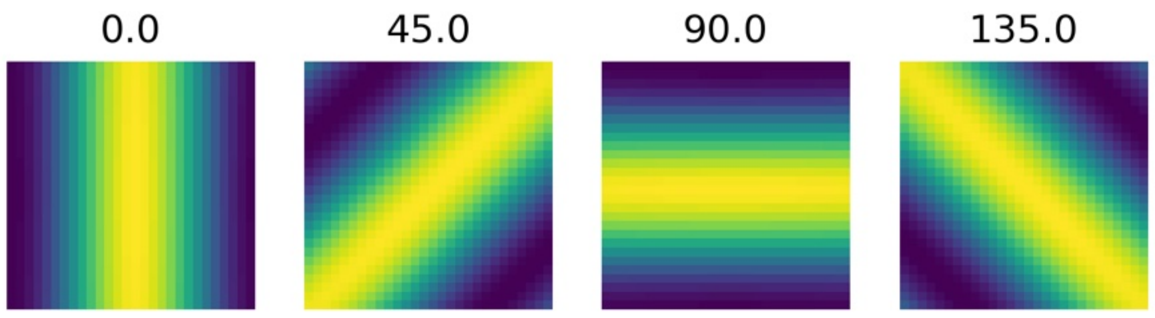


Figure 4.20: Possible inputs and their corresponding orientations for the rotation prediction toy problem.

4.2.2 Rotation prediction

The problem

One of the many properties of Gabor filters is that they are tuned to a well-defined and known orientation. By making use of this, we can set up a very simple classification task where we will try to predict the rotation of a given 2D sinusoid using only our defined Gabors. To make it simpler we will work only with 4 possible rotations: 0, 45, 90, and 135 degrees respectively. Figure 4.20 presents the four different inputs with corresponding labels.

To this effect, our model will consist only of a Gabor layer with 4 outputs, meaning that we will be optimizing 4 different Gabors to solve this task, as shown in Figure 4.21. More specifically, we will present two cases: (a) optimizing all the parameters in Eq. 4.3 and (b) optimizing only the parameter θ. By taking advantage of their known orientation, we can get the final prediction by taking the spatial average of each output and choosing the orientation that corresponds to the higher one. This would be equivalent to choosing the highest mean activation, which makes a lot of sense considering that a Gabor filter will produce greater activations if it is more aligned with the input sinusoid, thus giving us a good estimate of the input's orientation. It could be interesting to experiment with the $L2$ norm instead of the mean as a summarizing function.

Known solution

Having said this, we expect that each of the Gabors aligns itself with a different orientation, which would solve the problem with a 100% accuracy. The orientation of all the Gabors will be initialized at 0 degrees to assess their capability to choose different orientations in order to minimize the loss. Because the labels are encoded as $(0, 1, 2, 3)$, the maximum activation has to be on the filter of the same index to minimize the loss so, even though the filters are initialized at the same angle, there is only one configuration in which the loss is minimized, i.e. the first filter must be aligned with 0°, the second must be aligned with 45° and so on.

An interesting note would be that usual deep learning approaches would flatten or average the Gabor's outputs and put them through a dense layer in order to predict the input's class, but this would make it harder to disentangle the contribution of the Gabors and the dense layer. By following our approach we know that everything is performed

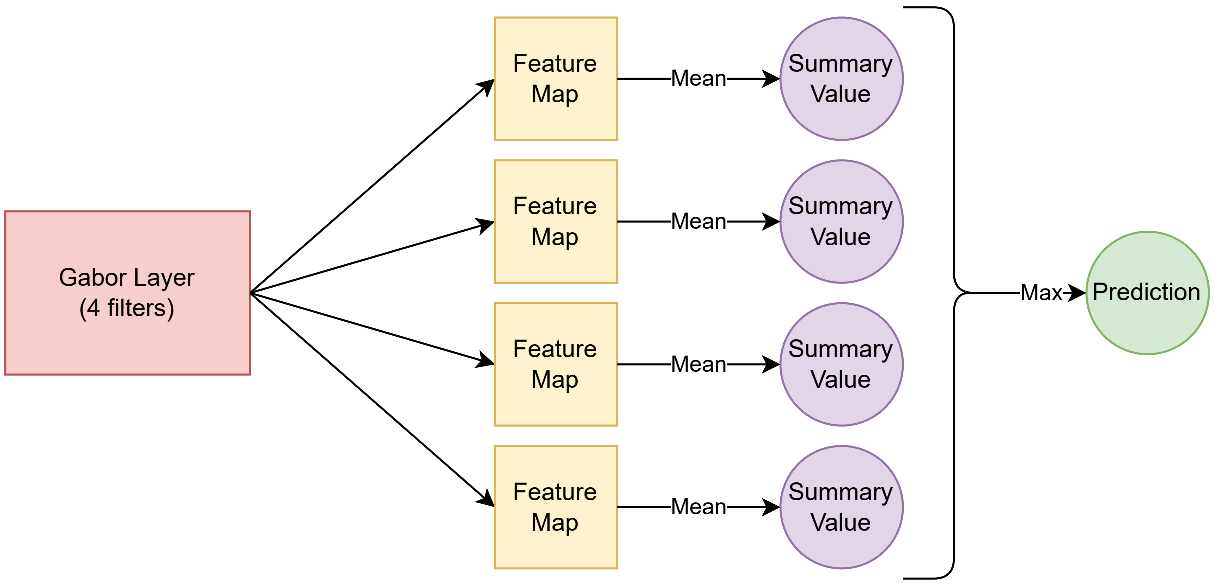


Figure 4.21: Schematic representation of the models used in Section 4.2.2. It consists of a first layer with 4 Gabor filters whose outputs are summarized by taking their spatial mean. The final prediction is obtained by taking the index of the maximum summary value.

only by the Gabor filters.

Obtaining the solution

Taking into consideration our findings from Section 4.2.1, we normalize the energy of the Gabor filters instead of their volume. It's interesting to note that this task, Rotation prediction, doesn't have such an easy shortcut as we found before because the only way of solving this problem is to actually orient the trainable Gabors according to the four different classes available: 0, 45, 90 and 135 degrees. As said before, we trained two variants of our model: one (a) that can optimize all the parameters of the filters and one (b) that can only modify its orientation θ, whose results are presented in Figure 4.22.

When looking only into the values of the orientations, we see differences in the convergence speed and its stability, which is quicker in the more constrained scenario. Notice that, this same behavior happened in the reconstruction task, where having more trainable parameters besides σ led to some plateaus during training (Figures 4.15 and 4.17). It comes to our attention that in scenario (a) the values of θ don't exactly align with the expected values. This can be attributed to the fact that the "effective rotation" of the filter depends on both θ and α, so the model is able to align the filters using both angles instead of only θ. This leads to interesting differences in the final trained filters, shown in Figures 4.23 and 4.24.

Here we see that when all the parameters are free to change, the filters don't only align with the desired orientations but change their widths and frequencies, resulting in a considerably lower loss value ($\mathcal{L} = 6.5 \cdot 10^{-5}$ when optimizing all the parameters vs $\mathcal{L} = 1.25$ when optimizing only θ). At first, this difference can be surprising but can be given an explanation taking into account the loss function used to train this model. The usual approach when working on a classification problem within the deep learning framework is to minimize the *cross-entropy* function. This function is not only concerned about the correct class having the highest probability, it also enforces that the other classes

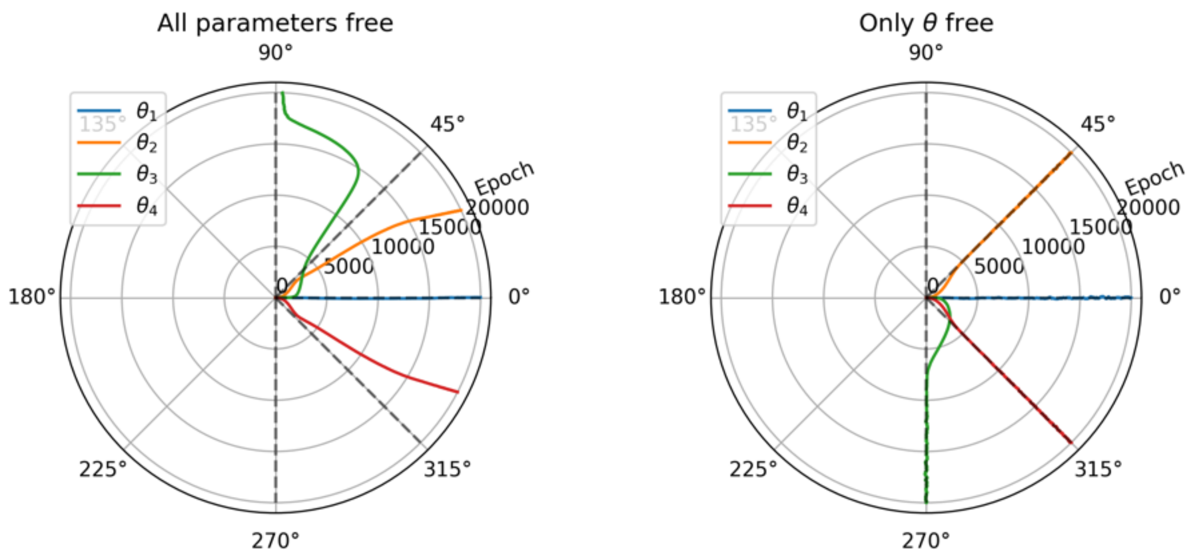


Figure 4.22: Evolution of the four different θ when leaving all the Gabor parameters free (left) and when constrained so that only θ is free to change (right).

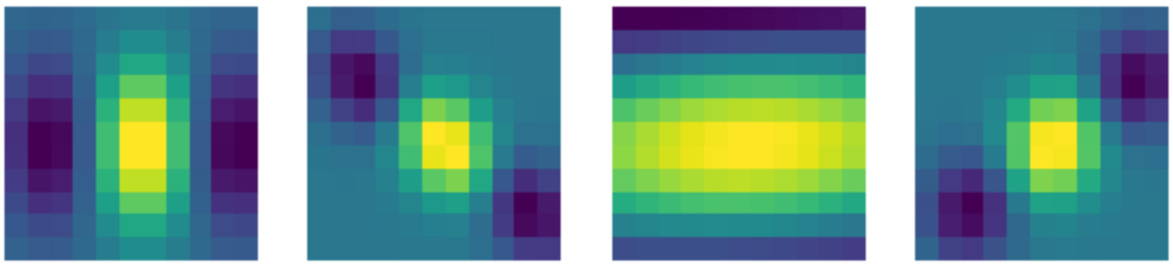


Figure 4.23: Final Gabor filters when all of its parameters are free to change during training.

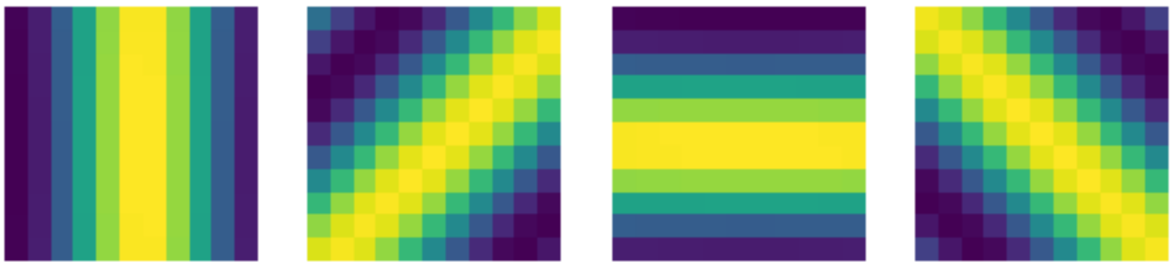


Figure 4.24: Final Gabor filters when only its orientation is free to change during training.

have as a low probability as possible. This might be troublesome within the task at hand because the classes we want to predict are not perpendicular between all of them, and thus when a filter is aligned with 0 deg, the filter aligned with 45 or 135 deg will still have a relatively high activation. This can be mitigated when we allow the filters to change also their widths and frequencies so that this loss can be minimized completely.

Consequences

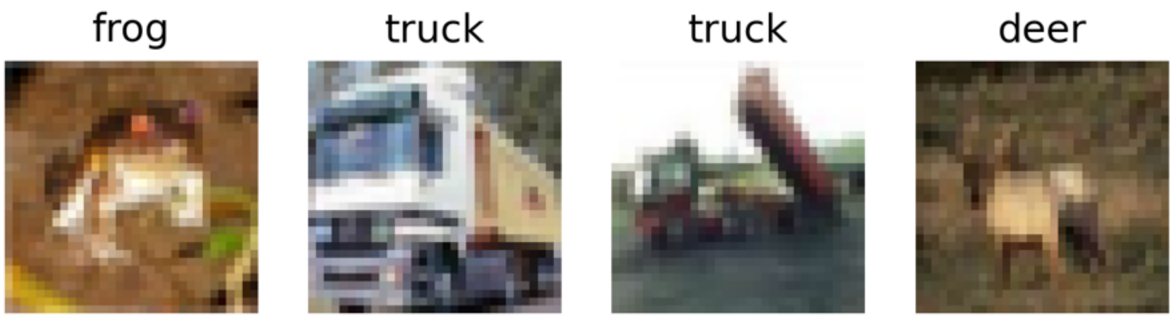
All in all, we were able to obtain the expected results in both toy problems and extracted a lot of useful information from them, while also identifying that these kinds of parametric filters may have some quirky behaviors during training because of the entanglement between their variables and the different shortcuts that could solve a given task. Some interesting findings of this experiment are that, sometimes, restricting the parameters that are allowed to change might be beneficial to enforce a specific behavior but utilizing more parameters may lead to a lower loss value. Also, normalizing the filters in different ways may push the solution outside of the available solution space, as happened when normalizing the Gaussian's volume, while normalizing the filters' energy provided the best results. As a side note, we found that cross-entropy breaks non-orthogonal problems in unusual ways due to its contrastive nature.

4.3 Classification

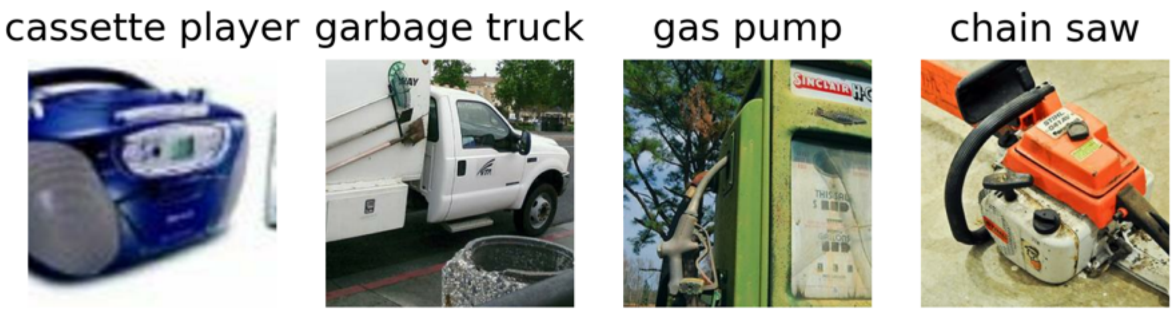
Problem setup

Moving away from toy problems, we would be interested in seeing if the use of parametric layers would improve the results obtained in a real classification problem. It is known [Krizhevsky et al., 2012] that the earlier layers of deep convolutional networks tend to learn Gabor-like features, so it would make sense to replace these early layers with parametric Gabor layers in order to reduce the number of parameters of our model and to speed up the training process in a similar fashion as in [Alekseev and Bobe, 2019]. A particular benefit of parametric filters is that the features they learn may be more general than those learnt by free-convolutions and as so, it may be possible to train a parametric model on a small dataset and then obtain good results on a different/bigger dataset resulting in a lot less training time. The idea supporting this is that free-convolutions might overfit more easily the data and thus would perform worse on different data.

With this objective in mind, we will tackle the well-known CIFAR10 classification problem [Krizhevsky, 2009]: 32x32 images in RGB representing 10 different classes. The approach will be to establish a baseline using a usual convolutional neural network and then try to improve upon it with a network based on parametric layers, taking into special consideration the difference in the number of trainable parameters. Once our models are trained on this small dataset, we will perform transfer learning to ImageNette [Howard, 2019], a subset of ImageNet, to see how they behave on 256x256 images which are indeed harder to classify. Keep in mind that this requires re-training the final dense layer because the classes are different, but it should favor the model that is able to extract the most useful and general features. These two stages would give us some insight into the advantage of using parametric filters for obtaining better results overall and its implications with regard to transfer learning. Some data samples from both datasets are shown in



(a) CIFAR10



(b) ImageNette

Figure 4.25: First four images of (a) CIFAR10 and (b) ImageNette. The most notable difference is that CIFAR10 images are 32x32 while ImageNette's are 256x256 while also being harder to classify.

Figure 4.25. It's clear that ImageNette images have a higher resolution than the ones in CIFAR10.

Within this task, we can compare parametric and non-parametric models in two different ways: (1) choosing the same architecture for both and comparing their performances with respect to the number of trainable parameters of each, and (2) adjusting different architectures but keeping the number of trainable parameters as close as possible by increasing the number of filters in the parametric model. (1) Has interest because it represents a 1-to-1 mapping on architectures but may be biased towards the non-parametric model because it doesn't have any restriction to its weights and could obtain better results. On the other hand, (2) is interesting because it compares models with presumably the same expressive capabilities, but may be biased towards the parametric model because increasing its number of filters will increase the width of the final dense layer, making it harder to attribute the improvement of performance to the parametric layer.

Experimental setup

The models used for this task (1) will be composed of a convolutional layer (either a parametric Gabor or a free-convolution) with 512 (11×11) filters, a global average pooling, and a dense layer at the end, making it the most simple model possible. In case (2) we will train a non-parametric model with 64 (11×11) filters as well. It can be seen in Figure 4.26. The models are trained with a learning rate of $3 \cdot 10^{-3}$ and a batch size of 128 for 500 epochs on CIFAR10 and a batch size of 64 for 500 epochs on ImageNette.

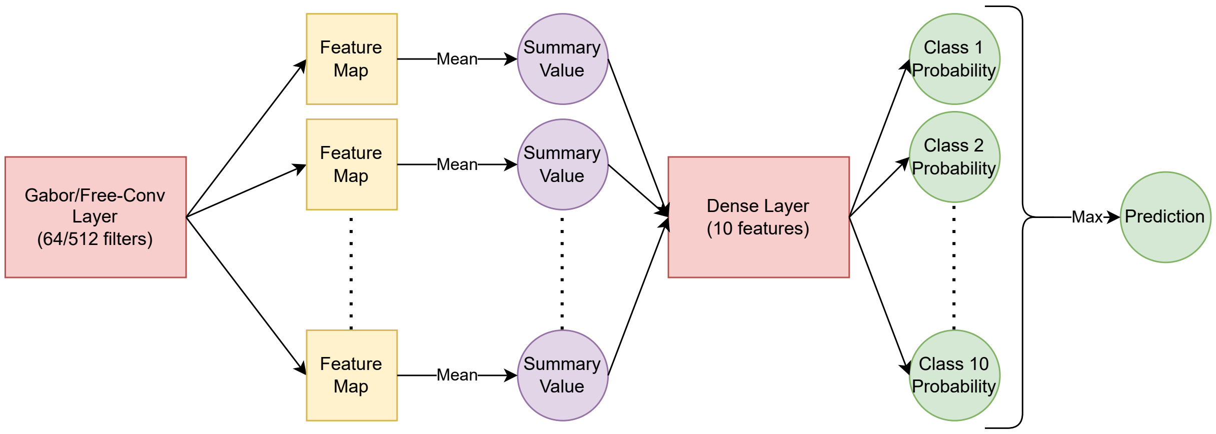


Figure 4.26: Representation of the models used to solve the classification task. For (1), we will be training a parametric and a non-parametric model with 512 filters, while for (2) we will train a non-parametric model with 64 filters. All the features extracted by the first parametric or non-parametric layer are summarized by taking the spatial mean and fed into a dense layer that outputs the probability of an image pertaining to one of the ten classes available in the dataset. When transferring the models from CIFAR10 into ImageNette, we will leave the first layers' weights fixed and modify the Dense layer to adapt to the new classes.

Keep in mind that the Gabor layer to be used has $7 \times 512 \times 3 + 512 = 10752$ parameters, while the free-convolutions have $11 \times 11 \times 3 \times 512 + 512 = 186.368$ and $11 \times 11 \times 3 \times 64 + 64 = 23.296$ for the 512 and 64 filters cases respectively.

4.4 Image Quality Assessment

Finally, we are going to face the main problem of this work: Image Quality Assessment (IQA). The explanation as to why this is our main problem is dual. First of all, the inspiration for this project came from a network built to serve as an IQA metric, so our final objective should be to compare with their result. But secondly, the three functional forms chosen are heavily inspired by visual science, and implementing them in a task that tries to resemble human behavior as much as possible seems like a very fitting task for these tools.

When working on a problem of regression or classification, our objective is usually to build a model that can take something as an input and then produces an output that can be, for example, its class, the location of an object in an image, a segmentation mask, etc. In an IQA task, we will deviate slightly from this approach and aim to build a model that takes an image as input and transforms it into a different space where the concept of distance is more aligned with the human concept of distance or, in other words, to a different space where the concept of distance is more correlated with human perception. We want to build a model that acts as a *perceptual metric*.

To this effect, IQA datasets are built like this: a not too big set of high quality images are selected as base or *reference* images. Then, a set of known transformations (brightness, noise, etc.) are applied to them in order to obtain different distortions of each image. Finally, a big group of humans is asked about how different the references and distorted images look to them, and this is taken as the *Mean Opinion Score* or MOS.

Whereas in other machine learning tasks a sample of data would consist of a 2-tuple (Features, Desired Output), in an IQA task we will consider a sample of data a 3-tuple (Reference image, Distorted image, MOS). With this in mind, our model f will be trained in order to maximize the correlation between the distances calculated on the transformed domain and the Mean Opinion Score. The corresponding loss function is laid out in Eq. 4.9, where x_o and x_d correspond to the reference and the distorted image respectively and ρ represents the Pearson correlation:

$$\mathcal{L} = \rho\left(||f(x_o) - f(x_d)||_2\,, MOS\right) \tag{4.9}$$

4.4.1 Perceptual distance

Most *humaness* tests used to evaluate artificial models rely on measuring differences between images (i.e. psychophysical tests [Graham, 1989, Regan, 1991] or correlation with human perception on image quality tasks). This *perceptual decisions will be made on the basis of the information available in the response space and not the input space.* The idea is to use a model to transform the images from the original domain into a response space and calculate the Euclidean distance between them in this transformed space. This distance is known by the name of *Perceptual Distance*, d_p, and can be written as in Eq. 4.10, where x^n represents the transformed version of the image x^0. Ideally, *a good perceptual distance mimics the human rating of similarity between two stimuli with high accuracy* [Hepburn et al., 2022].

$$d_p^2(x_A^0, x_B^0) = |x_B^n - x_A^n|_2^2 = \left(x_B^n - x_A^n\right)^T \left(x_B^n - x_A^n\right) = \Delta {x^n}^T \Delta x^n \tag{4.10}$$

It's very important to keep in mind that the Euclidean distance calculated on the transformed domain can be very different from that calculated on the original domain. It can be seen that it is, in fact, a non-Euclidean distance (in the original space) where the Jacobian of the model, $\nabla_x S$, acts as a non-Euclidean metric in Eq. 4.11 [Martinez-Garcia et al., 2018]:

$$d_p^2(x_A^0, x_B^0) = \Delta {x^0}^T \nabla_x S(x_A^0)^T \nabla_x S(x_A^0) \Delta x^0 \tag{4.11}$$

4.4.2 Data

There are quite IQA datasets in the wild, but we are going to utilize three of them: TID2008 [Ponomarenko et al., 2009], TID2013 [Ponomarenko et al., 2015], and KADID10K [Lin et al., 2019].

In our experiments, TID2008 will be used for training our model while TID2013 and KADID10K will act as validation and test sets respectively. This is the case because TID2008 and TID2013 share the same reference images but have different distortions, so validating with TID2013 allows us to assess the distortion generalization capabilities of our model. On the other hand, KADID10K is chosen as a test set because it has different reference and distorted images, so it can be considered as a good approximation to the image and distortion generalization capabilities of the model when deployed in the real world. Samples from TID and KADID are shown in Figures 4.27 and 4.28. Keep in mind that we will be using the smallest dataset to train our model, which is also interesting on its own because being able to obtain good generalization performances with as little

Table 4.1: Statistics of the datasets to be used: TID2008, TID2013, and KADID10K. TID2013 employs the same reference images as TID2008 but includes new distortions and intensities, so it is a good indicator of the generalization capabilities of the model to new distortions. KADID10K, on the other hand, utilizes a different set of both reference images and distortions, so it is a good proxy for the generalization capabilities of our models when deployed in the real world.

Dataset	Samples	# Refs	# Dists	# Intensities	Max. ρ
TID2008	1700	25	17	4	0.86
TID2013	3000	25	24	5	0.83
KADID10K	10125	81	25	5	0.78

training data as possible is always a thing to look out for. Some useful information about the dataset sizes is provided in Table 4.1.

Maximum attainable performance

A particularity of these datasets is that they provide the mean, μ, and the standard deviation, σ, of the measured Mean Opinion Scores and, by making use of these, we can calculate the maximum attainable correlation by our model through a basic Monte Carlo experiment: one can sample data from a Normal distribution with the given μ and σ to generate "sampled" experiment executions. Then, it's easy to calculate the correlation between each sampled experiment and the given MOS, obtaining a correlation for each repetition. By repeating the simulation N times, we obtain N values of correlation with respect to the given MOS and then obtain the maximum attainable correlation by taking their mean. The results obtained for each dataset are shown in the rightmost column of Table 4.1.

Our results are quite surprising because TID2008 and TID2013 have a maximum attainable correlation of $\rho = 0.86$ and $\rho = 0.83$ respectively, while KADID10K's is a bit lower at $\rho = 0.78$. This can be attributed to the fact that KADID10K is a much bigger dataset and, because of that, they couldn't record as many human evaluations as they needed to reduce the disparity of the measures. Adding to this, their distortions include some higher-level distortions that are usually harder to assess and could result in less precise measurements. It is to be taken into consideration that the authors of these datasets make the assumption that the evaluations for each (Reference, Distorted) pair follow a Normal distribution, but as we only see the summary values, we can't confirm this. If the data wasn't normally distributed but, maybe, Laplacian, the maximum attainable correlation may be different.

With this in mind, we can consider a good result for any model that obtains above the corresponding maximum correlations on each dataset.

4.4.3 Model

As we reviewed in Section 3.2.3, PerceptNet's architecture is chosen in order to perform a specific set of transformations, but its weights did not converge to it. We will take all the parametric layers we defined in Section 4.1 and place them at specific locations in the architecture so that they perform their given task. Getting into the details: (1) all the

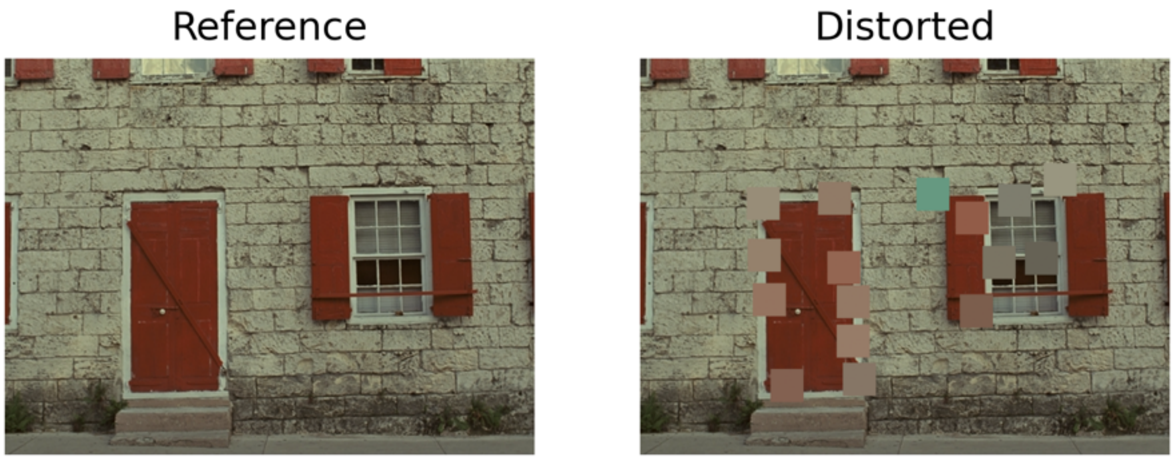


Figure 4.27: A Reference and Distored pair of images from TID2008 with a corresponding MOS of 2.5143.

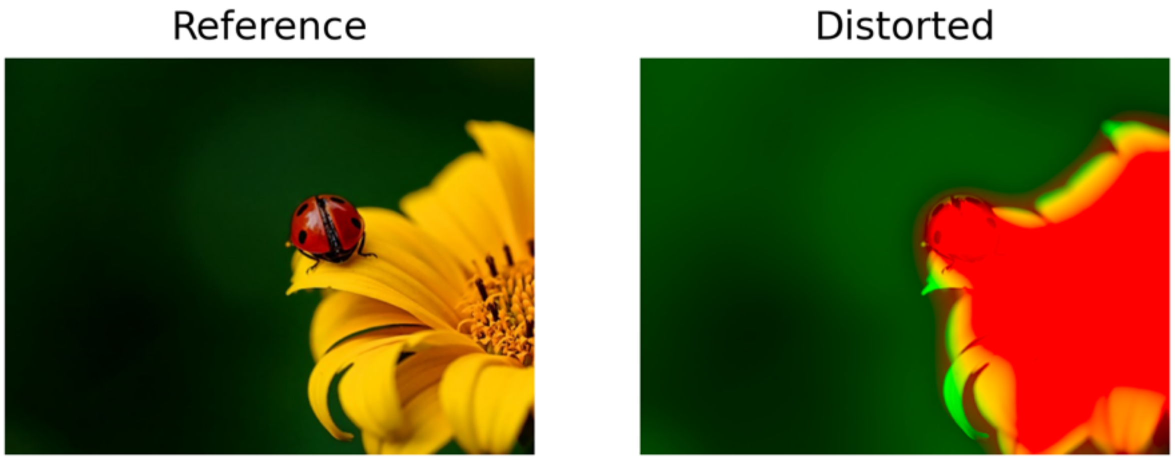


Figure 4.28: A Reference and Distored pair of images from KADID10K with a corresponding MOS of 9.

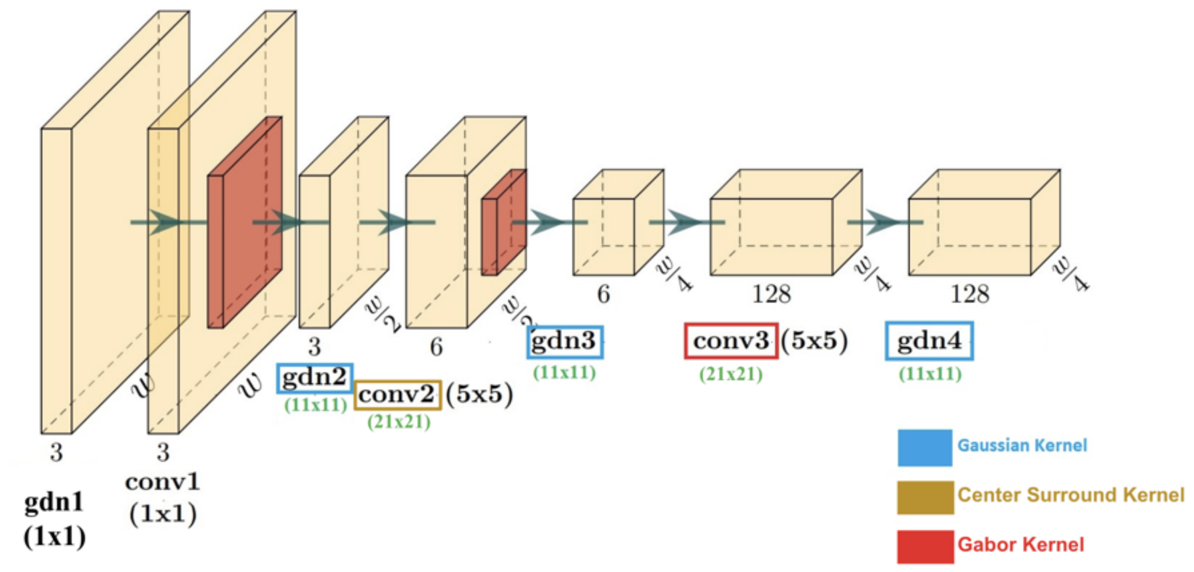


Figure 4.29: Parametric version of PerceptNet [Hepburn et al., 2020]. Spatial Gaussians replace the convolutional kernels of the Divisive Normalization to allow for capturing more information about the surroundings. Filters in the retina-LGN section are constrained to be center-surround cells and the ones in the V1 section are parametrized by Gabor functions. Some of the kernels' spatial size has been increased. The new sizes are shown in green.

Divisive Normalizations (*gdn2-4*) but the first one will have their convolutional kernel changed into a parametric spatial Gaussian that will allow it to take into consideration the neighbor pixels in the operation. (2) The second convolutional layer (*conv2*), whose task is resembling the retina-LGN section in the visual pathway, will be changed into a parametric center-surround, and (3) the final convolutional layer (*conv3*), acting as the V1 section, will be changed by a parametric Gabor layer. All of these choices are motivated by the literature we went through in Section 3.3 and the design choices of PerceptNet and, by performing these changes, we no longer train all the weights in the kernels separately and hope that they end up performing the transformations that we want. Now we impose the form of the computations that we want to happen and optimize only the parameters of those transformations, which makes sense if what we want to achieve is a vision model that performs as human-like as possible. These changes are introduced in Figure 4.29, where we show which layers are changed by what parametric layers and the sizes of the new kernels as well.

Apart from training the fully parametric model, we will also perform an ablation study of the three changes [(1),(2),(3)] with the objective of studying which modifications affect more our results and which are going to be the most important changes in the architecture.

Another important part of our study is showing how the number of parameters in the model is reduced when introducing parametric layers. We have to keep in mind that in the original architecture, the Divisive Normalization layers uses $1x1$ convolutions which have a very minimum number of parameters thus when using spatial Gaussians, we will surely introduce more parameters. As an example, the final Divisive Normalization (*gdn4*) in the original model has 16.512 parameters, but when parametrizing it with Gaussians we need 32.896. The intuition behind this is that a $1x1$ kernel only has one weight per input channel, while we need to build a parametric filter for each input channel and the

Table 4.2: Number of parameters of each of PerceptNet's parametric variations.

Model variation	# Parameters	% to Original
Original	36.368	1.0
GDNGaussian (1)	52.660	1.45
CenterSurround (2)	35.966	0.99
GaborLast (3)	22.544	0.62
GDNGaussian + CenterSurround (1+2)	52.258	1.44
GDNGaussian + GaborLast (1+3)	38.845	1.07
CenterSurround + GaborLast (2+3)	22.014	0.61
Full (1+2+3)	38.306	1.05
Original (Same Size)	2.334.824	64.2

Gaussians have two parameters: σ and A, doubling the number of parameters. This is negligible when considering that the parametric filters don't increase their number of parameters when increasing the kernel size, but the free-convolution does increase so, if we wanted to use Divisive Normalizations whose kernels were the same size as the Gaussians (11x), we would have almost 2M parameters, proving to be an impressive reduction in this case. Table 4.2 summarizes the number of parameters each of the parametric PerceptNet's variations have and its relative value with respect to the original non-parametric model. We have also added the number of parameters of a free-convolutional model with the same kernel sizes as those in the parametric models, which goes well above 2M parameters but we won't be training this model, its shown only for demonstration purposes. Be aware that the most reduction in the number of parameters comes from substituting the last free-convolution with a Gabor layer (even though the kernel size of the Gabor is more than 4 times bigger). All the models have been trained for 500 epochs with a batch size of 64 and a learning rate of $3 \cdot 10^{-4}$ to ensure that the differences in performance could be attributed to the changes in the architecture.

Chapter 5

Results

This Chapter will go through the results obtained for the experiments layed out in Sections 4.3 and 4.4, which corresponds to our objectives 3a and 3b. We will be comparing the performance of the parametric and non-parametric models for a classification and an image quality task by checking their final performances and their training dynamics, while also paying attention to some details found when training deep learning models including functional forms.

5.1 Classification

The accuracy obtained for both parametric and non-parametric models on CIFAR10 and ImageNette is shown in Table 5.1, where it can be seen that the best performance on CIFAR10 is obtained by the parametric model which, in fact, has more than **11 times** fewer parameters than the non-parametric model with the same number of filters, case (1), and **1.5 times** fewer parameters than the non-parametric model with the closest amount of parameters, case (2). By taking a look at [Jeevan and Sethi, 2021] we see that it obtains even better performance than different variations of the Visual Transformer [Dosovitskiy et al., 2021] with a number of parameters ranging from 400k-8M parameters[1].

More interesting than the raw accuracy number itself are the training dynamics of the three models, shown in Figure 5.1. It's clear from this Figure that the non-parametric model with higher number of filters (1) is much more prone to overfitting the training data than the parametric model, and the parametric model could even be trained further to obtain higher accuracy. The reduced non-parametric model doesn't show signs of overfitting but lacks the expressive capability to attain a result as good as the other models.

[1]This has to be taken with a grain of salt because our non-parametric model also obtains better performances, but still is a nice comparison to take into account.

Table 5.1: Accuracies in CIFAR10 and ImageNette for the parametric and non-parametric model.

Model variation	# Parameters	CIFAR10 (Acc.)	ImageNette (Acc.)
Non-Parametric	23,946	60.1	46.6
Non-Parametric (512)	191.498	65.0	**52.0**
Parametric	**16.394**	**67.0**	51.0

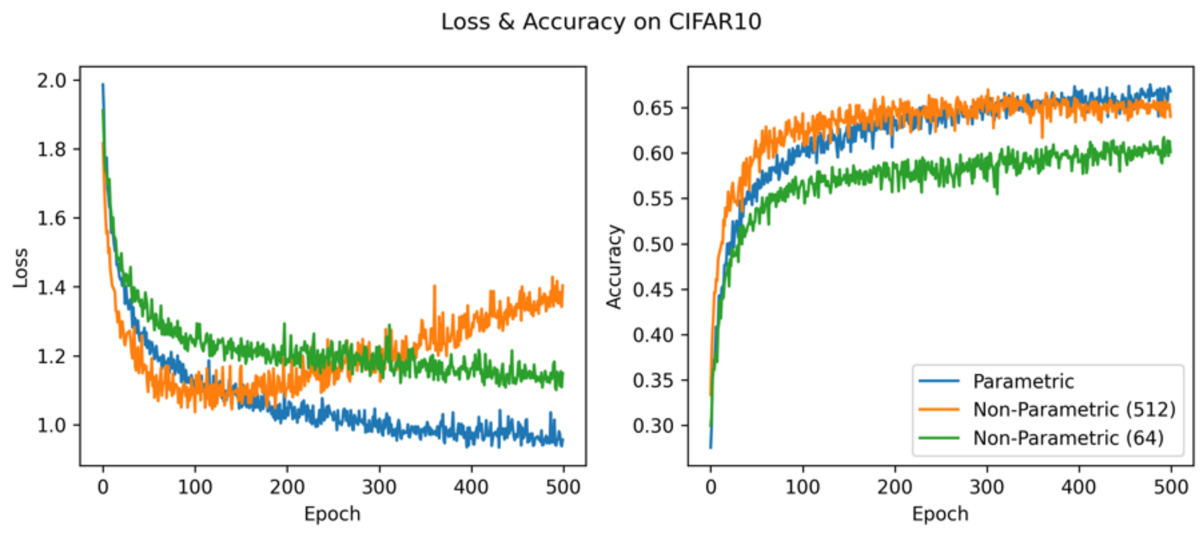


Figure 5.1: Training dynamics of the parametric (blue) and non-parametric (orange) models in a classification task on the validation set of CIFAR10. It's clear that the non-parametric model quickly overfits the training distribution while the parametric model doesn't, indicating its robustness with high learning rates.

This makes sense because the parametrization of the kernels can be seen as a form of regularization, which is a technique often used to overcome overfitting. These results can serve as a confirmation, although it would require training on more datasets and with bigger models, that parametric convolutional kernels can be trained at higher learning rates than free-convolutions, reducing training times and therefore reducing costs and expenses. On a side note, our results are much tighter than those shown in [Alekseev and Bobe, 2019], where they show that the parametric Gabor filters obtain a better performance since the beginning of the training, but this can be attributed to their initialization, which is based on the parameters of a specific filter bank while our parameters are initialized randomly within a set of ranges that make sense.

Regarding its transfer learning capabilities, the non-parametric model obtained a slightly better accuracy but when looking at the losses of the models with same amount of filters (Figure 5.2) the parametric model obtained better values at every step during training.

To wrap up, we can see some of the most important[2] convolutional filters learnt by the models in Figure 5.3. The filters learnt by the free-convolution don't have any particular shapes or characteristics that could be analyzed, while the parametric Gabors show different widths, orientations, and frequencies, suggesting a good variety of features to be extracted. The repetition of a very sharp filter seems interesting but we didn't find any specific explanation for it. Another difference with respect to free-convolutions is that we could choose to remove the filters that have too similar properties because their parameters are easily interpretable, something that is much harder to do with free-convolutions, where we see a lot of repetitions of the same kind of filter which doesn't have much meaning.

[2]We choose the most important filters by looking at their norm (in the case of free-convolutions) or at the parameter $\hat{A}$ for parametric Gabors.

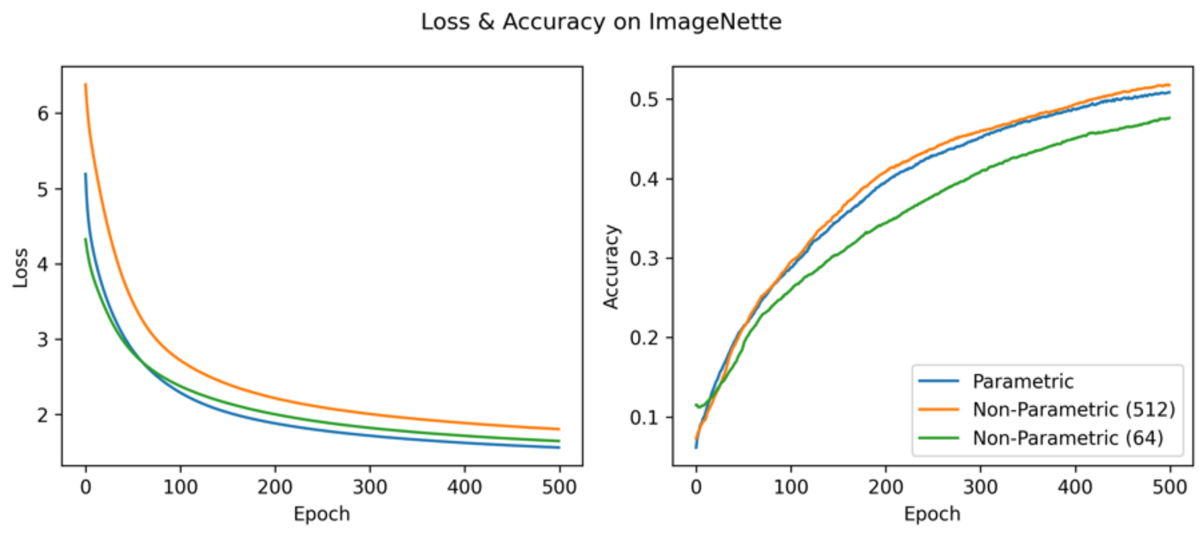


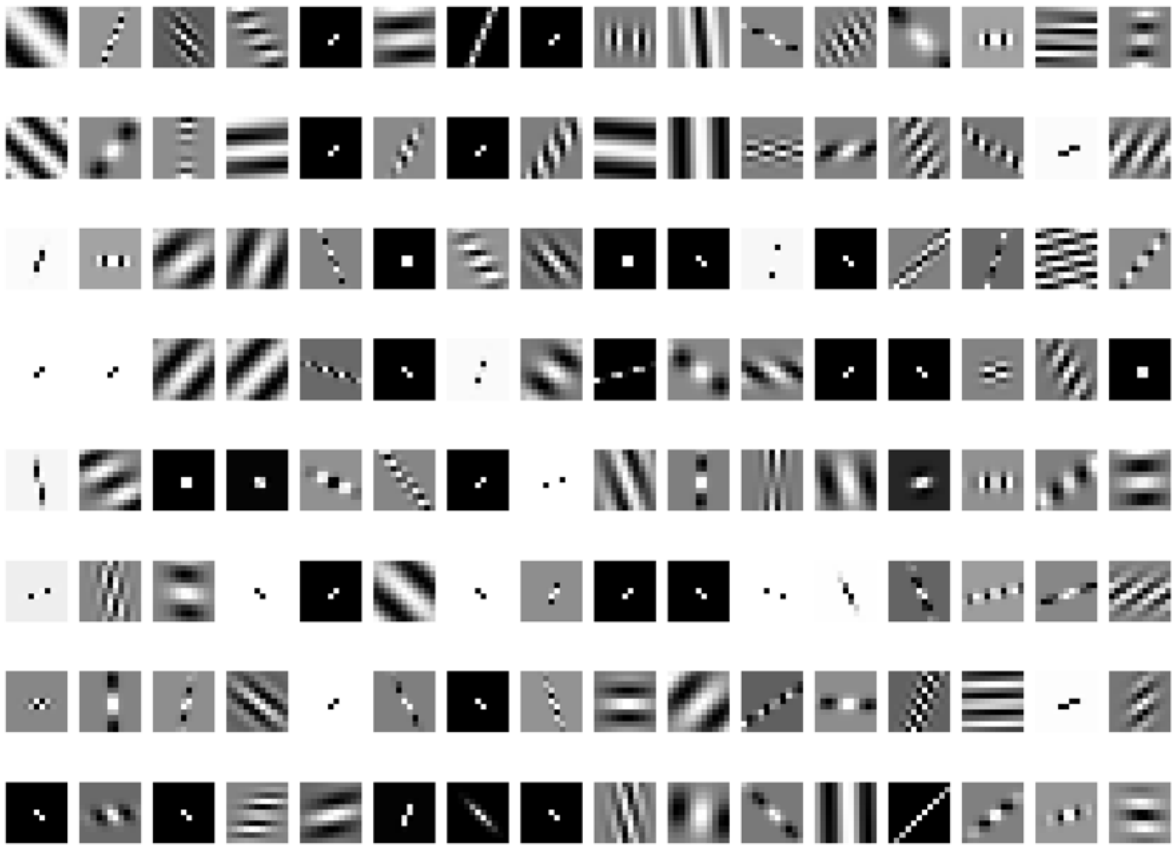
Figure 5.2: Training dynamics of the parametric (blue) and non-parametric (orange & green) models when performing transfer learning from CIFAR10 into ImageNette. The non-parametric model obtains a slightly better accuracy, but the parametric model obtains lower loss across all training epochs.

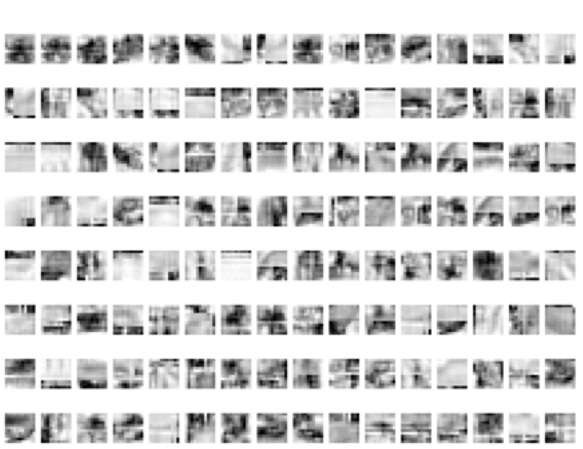
5.2 Image Quality Assessment

Pearson correlation on the different datasets for all the model variations are shown in Table 5.2. Spearman correlations are also shown in Table 5.3 but only for completion, as we will only focus on Pearson correlation.

The first and most important result is that the non-parametric baseline is surpassed in performance by almost every parametric variant studied, even by those with lower number of trainable parameters. We also see that almost all of the parametric models obtain correlations, on all three datasets, higher than the maximum attainable correlation calculated in Section 4.4.2, allowing us to say that they have good enough generalization capabilities because they pass the benchmark. On a different note, we see noticeable differences in results obtained by models that employ different forms of normalization (as was expected from Section 4.2) as well as slight differences in training dynamics when using (or not using) biases within the parametric layers. This may indicate that: (1) introducing biologically inspired functional forms into our model leads to an increase in performance, and (2) using functional forms in conjunction with the more common deep learning approaches may require more investigation related to their training dynamics if we want to attain their maximum potential. It is natural to think that the deep learning framework has been greatly optimized to work with non-parametric functions and, as such, it could require further tweaking to work best with restricted or parametric forms.

An important subtlety we found out when performing the experiments is that, for the Divisive Normalization to work properly, its kernel must be clipped to allow only values $H_{ij} >= 0$. This was not a problem when working with Gaussians because they are positive definite, but it was a problem when testing the free-convolution implementation. We knew we had to constrain the denominator of the Divisive Normalization because of the square root and so we applied a ReLU non-linearity to it. Turns out this approach induces a lot of instabilities during training that don't happen when clipping the kernel values directly, as is shown in Figure 5.4. Because the convolution is applied to the input squared, we can


(a) Parametric


(b) Non-Parametric (512)

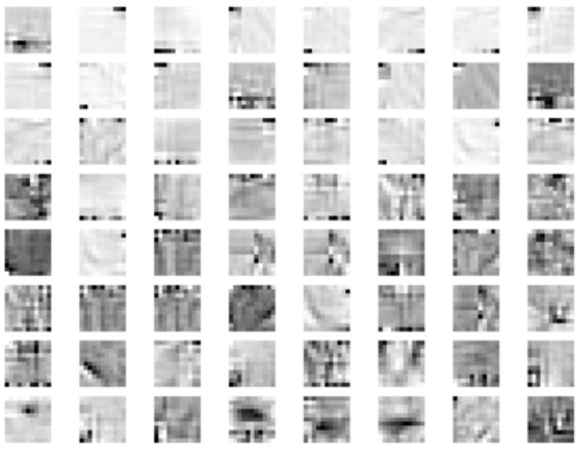

(c) Non-Parametric (64)

Figure 5.3: Final learnt filters by the parametric (a) and non-parametric (b) & (c) models. We see that the filters learnt by the free-convolution don't have specific shapes and show a lot of repetitions while the parametric filters are a lot easier to interpret, have defined shapes, and are show more variety.

Table 5.2: Pearson correlation for different model configurations in the different perceptual databases considered.

Name	TID2008	TID2013	KADID10K
Baseline	-0.92	-0.90	-0.76
GaussianGDN + NormEnergy (1)	-0.93	-0.90	-0.82
CenterSurround_NormEnergyProb (2)	**-0.94**	-0.91	-0.76
CenterSurround (2)	**-0.94**	**-0.92**	-0.65
GaborLast_NormEnergy (3)	**-0.94**	-0.91	**-0.83**
GDNGaussian + CenterSurround (1+2)	-0.91	-0.89	-0.78
GaussianGDN + GaborLast (1+3)	**-0.94**	-0.91	-0.82
CenterSurroundK + GaborLast (2+3)	-0.93	-0.90	-0.79
GDNGaussian + CenterSurround + GaborLast (1+2+3)	-0.93	-0.90	-0.79

Table 5.3: Spearman correlation for different model configurations in the different perceptual databases considered.

Name	TID2008	TID2013	KADID10K
Baseline	-0.92	-0.89	-0.84
GaussianGDN_NormEnergy (1)	-0.92	-0.89	**-0.86**
CenterSurround_NormEnergyProb (2)	**-0.94**	-0.90	-0.83
CenterSurround_K (2)	**-0.94**	**-0.91**	-0.79
GaborLast_NormEnergy (3)	**-0.94**	-0.90	**-0.86**
GDNGaussian + CenterSurround_Bias (1+2)	-0.91	-0.88	-0.84
GaussianGDN + GaborLast (1+3)	**-0.94**	-0.89	**-0.86**
CenterSurroundK + GaborLast (2+3)	-0.93	-0.89	-0.83
GDNGaussian + CenterSurround + GaborLast (1+2+3)	-0.93	-0.89	-0.84

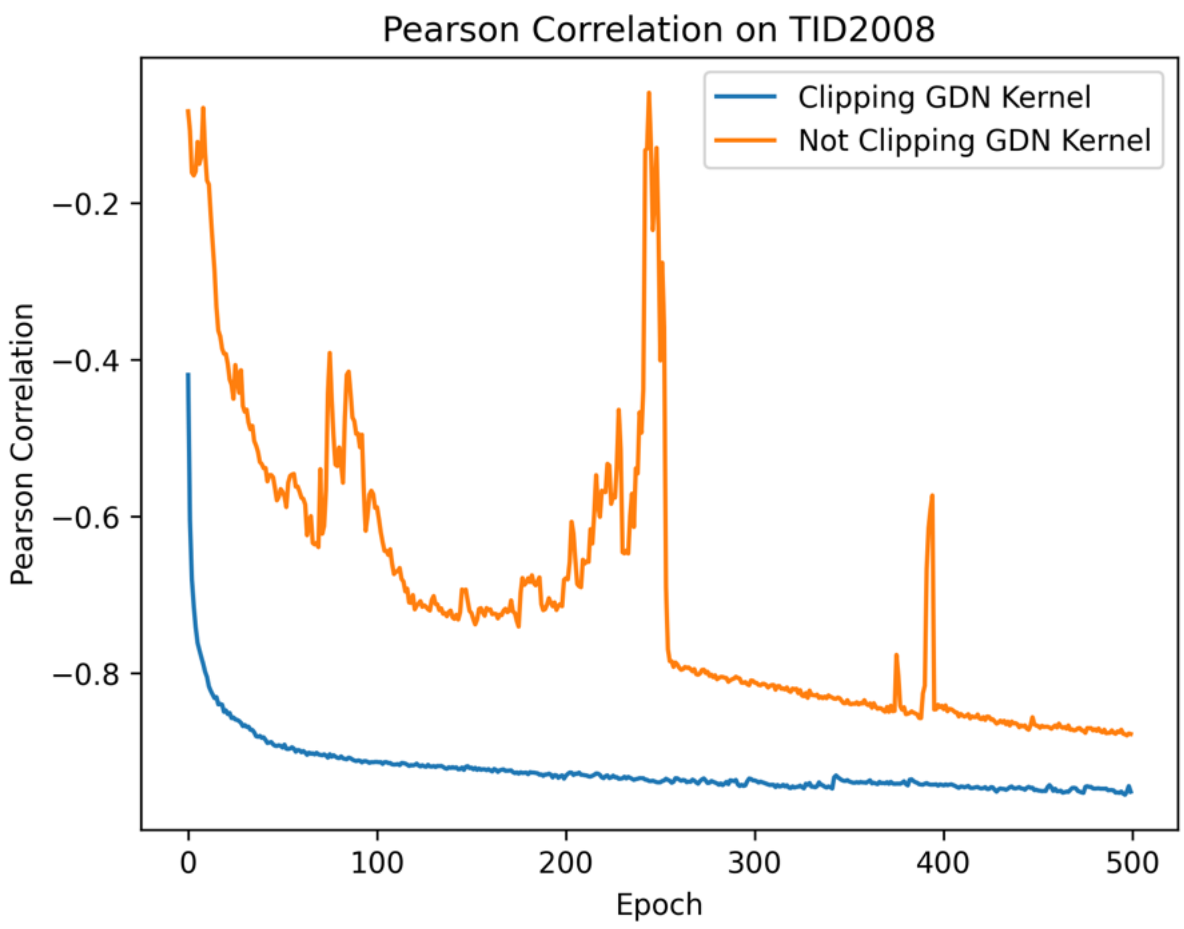


Figure 5.4: Training dynamics of the Baseline PerceptNet model while clipping the weights of the Divisive Normalization (blue) kernel and not clipping them but applying a ReLU non-linearity to the output of the convolution (orange).

enforce an all-positive output by clipping the values of the kernel, which resulted in a very smooth training that yielded much better results indeed. We can blame these instabilities on the ReLU producing a lot of 0s that didn't behave properly with the division operation (while still having negative numbers in the kernels), whereas by clipping the values of its kernel, the weights quickly move away from negative numbers and are better-behaved during training.

On a different note, we found that modeling the center-surround cell with $\sigma_2 = k\sigma_1$ produced better results than treating both σ_1 & σ_2 as independent parameters, probably due to the fact that they are indeed coupled and gradient descent may lead them to an incorrect optimum because it doesn't know that information if we don't introduce it ourselves. It is also noteworthy that the center-surround model obtained the best performance on TID, both 2008 (train) and 2013 (validation), but failed to generalize well to KADID10K until we normalized the energy of its filters, adding to our findings in Section 4.2. Touching briefly on it, when the filters were not normalized their outputs' magnitude were 20 times higher, resulting in very different outputs for slightly different inputs: a known source of poor generalization in machine learning models [LeCun et al., 2012].

Finishing up with the analysis of the results, we found that while it's common practice to add biases to convolutional layers, using a bias in our parametric Gabor layers introduced slight instabilities that the model had to recover from during training. These instabilities

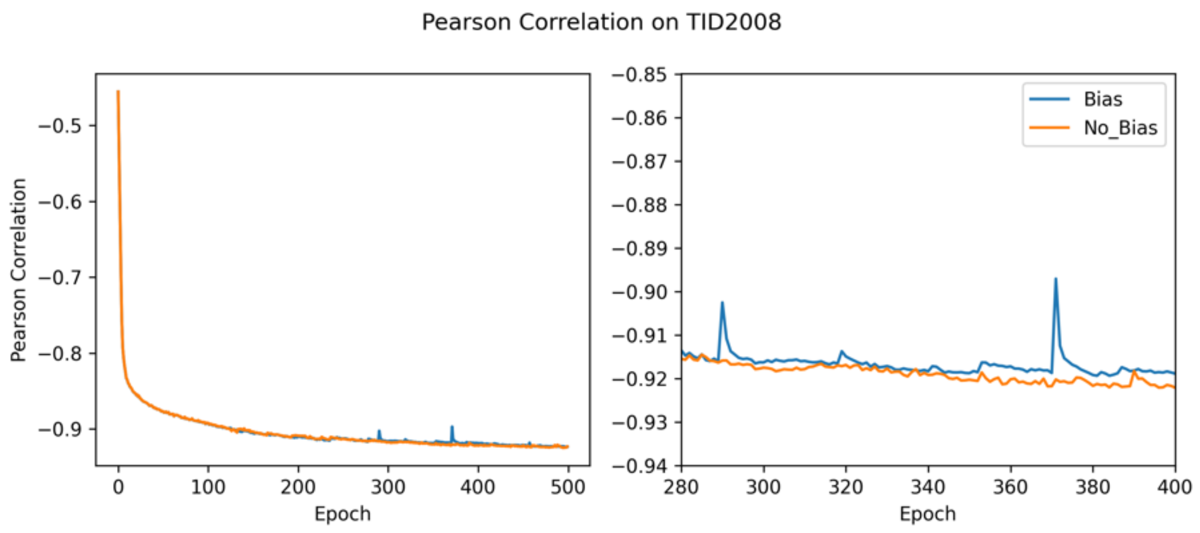


Figure 5.5: Training dynamics of the GaborLast with (blue) and without (orange) bias. We can observe that the use of bias increases the instabilities during training but doesn't affect greatly the performance of the model nonetheless.

arose in the form of sudden peaks in the loss function, as shown in Figure 5.5, but it's worth noting that the overall dynamics of the model weren't altered and their final performance was quite similar.

Summing up, we found the most success with the models that used Gabors and Gaussian Divisive Normalizations, as opposed to center-surround-based models, which performed slightly better on the training and validation sets but generalized worse to KADID10K. This is in line with [Evans et al., 2022], where they found that introducing Gabors into fully convolutional networks resulted in a performance boost while adding the Difference of Gaussian filters did not. Even a combination of both performed worse, which is still seen in our experiments.

Chapter 6

Conclusions & Future Work

Considering the technical objectives we laid out at the beginning of the project, we can confirm that this work has been a success because we were able to implement a set of parametric forms so that their parameters could be optimized through gradient descent, while the two toy problems that we designed were very useful both to test our implementations and extract a lot of information of the ins and outs of optimizing these kinds of functions within the conventional deep learning framework.

Building from these technicalities, we were also able to apply our functional forms to two real-world problems: classification and image quality assessment. In the first one, we showed the potential of restricting artificial neural networks with parametric forms by obtaining better performance with a parametric model than with a non-parametric model while also having a lot fewer parameters, even though we didn't find the same success when applying transfer learning into a different dataset, where the non-parametric model performed slightly better. Even if it wasn't our initial intention, we also showed that parametric models are more robust to overfitting and as such can be trained with higher learning rates, leading to shorter training times. Finally, we modified an already published model (PerceptNet) to impose a set of parametric operations inspired by statistics and biology, allowing us to obtain better results than the original model and fulfilling the original purpose of PerceptNet: applying a known set of transformations, which wasn't happening originally because the non-parametric convolutions weren't learning the intended operations.

All in all, we can consider that this work has been completed with great success and opens a very interesting path for future investigation by allowing us to constraint models in a flexible way that can be included as a drop-in replacement in artificial neural networks and has shown both a potential increase in performance and a notable reduction in the number of parameters.

As continuations and extensions of this work, we propose including the phase in the Gabors and implementing complex V1 cells, exploring the effect of introducing the coefficient in the Divisive Normalization, and investigating additional forms of normalizing the filters taking inspiration from already-used weight initialization techniques, exploring the implications of repeating filters (i.e. using the same filter for all the input channels) to reduce more the number of parameters while performing a stronger form of regularization, and looking into performing parameter-aware pruning, where we could make us of the known parametric forms to prune repeated filters and improve our models to reduce redundancy.

Bibliography

[Alekseev and Bobe, 2019] Alekseev, A. and Bobe, A. (2019). GaborNet: Gabor filters with learnable parameters in deep convolutional neural networks. arXiv:1904.13204 [cs, eess].

[Assirati et al., 2014] Assirati, L., Silva, N. R., Berton, L., Lopes, A. A., and Bruno, O. M. (2014). Performing edge detection by Difference of Gaussians using q-Gaussian kernels. *Journal of Physics: Conference Series*, 490:012020.

[Babaiee et al., 2021] Babaiee, Z., Hasani, R., Lechner, M., Rus, D., and Grosu, R. (2021). On-Off Center-Surround Receptive Fields for Accurate and Robust Image Classification. arXiv:2106.07091 [cs].

[Barlow, 1961] Barlow, H. B. (1961). Possible Principles Underlying the Transformations of Sensory Messages. In Rosenblith, W. A., editor, *Sensory Communication*, pages 216–234. The MIT Press.

[Bengio et al., 2016] Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., and Lin, Z. (2016). Towards Biologically Plausible Deep Learning. arXiv:1502.04156 [cs].

[Bergstra et al., 2011] Bergstra, J., Courville, A., and Bengio, Y. (2011). The Statistical Inefficiency of Sparse Coding for Images (or, One Gabor to Rule them All). arXiv:1109.6638 [cs].

[Calderón et al., 2003] Calderón, A., Roa Ovalle, S., and Victorino, J. (2003). Handwritten Digit Recognition using Convolutional Neural Networks and Gabor filters.

[Carandini et al., 2005] Carandini, M., Demb, J. B., Mante, V., Tolhurst, D. J., Dan, Y., Olshausen, B. A., Gallant, J. L., and Rust, N. C. (2005). Do We Know What the Early Visual System Does? *The Journal of Neuroscience*, 25(46):10577–10597.

[Carandini and Heeger, 1994] Carandini, M. and Heeger, D. J. (1994). Summation and division by neurons in primate visual cortex. *Science*, 264(5163):1333–1336.

[Carandini and Heeger, 2012] Carandini, M. and Heeger, D. J. (2012). Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51–62. Number: 1 Publisher: Nature Publishing Group.

[Cirincione et al., 2023] Cirincione, A., Verrier, R., Bic, A., Olaiya, S., DiCarlo, J. J., Udeigwe, L., and Marques, T. (2023). Implementing Divisive Normalization in CNNs Improves Robustness to Common Image Corruptions.

[DeWeese et al., 2003] DeWeese, M. R., Wehr, M., and Zador, A. M. (2003). Binary Spiking in Auditory Cortex. *The Journal of Neuroscience*, 23(21):7940–7949.

[Ding et al., 2020] Ding, K., Ma, K., Wang, S., and Simoncelli, E. P. (2020). Image Quality Assessment: Unifying Structure and Texture Similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1. arXiv:2004.07728 [cs].

[Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs] version: 2.

[Enroth-Cugell and Robson, 1966] Enroth-Cugell, C. and Robson, J. G. (1966). The contrast sensitivity of retinal ganglion cells of the cat. *The Journal of Physiology*, 187(3):517–552.

[Evans et al., 2022] Evans, B. D., Malhotra, G., and Bowers, J. S. (2022). Biological convolutions improve DNN robustness to noise and generalisation. *Neural Networks*, 148:96–110.

[Fukushima, 1975] Fukushima, K. (1975). Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3):121–136.

[Gabor, 1946] Gabor, D. (1946). Theory of communication. Part 1: The analysis of information. *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, 93(26):429–441. Publisher: IET Digital Library.

[Gerstner et al., 1996] Gerstner, W., Kempter, R., van Hemmen, J. L., and Wagner, H. (1996). A neuronal learning rule for sub-millisecond temporal coding. *Nature*, 383(6595):76–78. Number: 6595 Publisher: Nature Publishing Group.

[Gomez-Villa et al., 2020] Gomez-Villa, A., Martín, A., Vazquez-Corral, J., Bertalmío, M., and Malo, J. (2020). Color illusions also deceive CNNs for low-level vision tasks: Analysis and implications. *Vision Research*, 176:156–174.

[Graham, 1989] Graham, N. V. S. (1989). *Visual pattern analyzers.* Visual pattern analyzers. Oxford University Press, New York, NY, US. Pages: xvi, 646.

[Granlund, 1978] Granlund, G. H. (1978). In search of a general picture processing operator. *Computer Graphics and Image Processing*, 8(2):155–173.

[Graps, 1995] Graps, A. (1995). An Introduction to Wavelets.

[Gutmann and Hyvärinen, 2010] Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304. JMLR Workshop and Conference Proceedings. ISSN: 1938-7228.

[Hasani et al., 2019] Hasani, H., Soleymani, M., and Aghajan, H. (2019). Surround Modulation: A Bio-inspired Connectivity Structure for Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs].

[Hepburn et al., 2020] Hepburn, A., Laparra, V., Malo, J., McConville, R., and Santos-Rodriguez, R. (2020). PerceptNet: A Human Visual System Inspired Neural Network for Estimating Perceptual Distance. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 121–125. arXiv:1910.12548 [cs, eess, stat].

[Hepburn et al., 2022] Hepburn, A., Laparra, V., Santos-Rodriguez, R., Ballé, J., and Malo, J. (2022). On the relation between statistical learning and perceptual distances. arXiv:2106.04427 [cs, eess, q-bio].

[Hernández-Cámara et al., 2022] Hernández-Cámara, P., Laparra, V., and Malo, J. (2022). Neural Networks with Divisive normalization for image segmentation with application in cityscapes dataset. arXiv:2203.13558 [cs].

[Hernández-Cámara et al., 2023] Hernández-Cámara, P., Vila-Tomás, J., Laparra, V., and Malo, J. (2023). Analysis of Deep Image Quality Models. arXiv:2302.13345 [cs].

[Hinton, 2022] Hinton, G. (2022). The Forward-Forward Algorithm: Some Preliminary Investigations. arXiv:2212.13345 [cs].

[Howard, 2019] Howard, J. (2019). Imagenette.

[Hubel and Wiesel, 1962] Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154.2.

[Hyvärinen and Oja, 2000] Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430.

[Jeevan and Sethi, 2021] Jeevan, P. and Sethi, A. (2021). Vision Xformers: Efficient Attention for Image Classification. arXiv:2107.02239 [cs] version: 4.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. arXiv:1312.6114 [cs, stat].

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. page 60.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[Kumar et al., 2022] Kumar, M., Houlsby, N., Kalchbrenner, N., and Cubuk, E. D. (2022). Do better ImageNet classifiers assess perceptual similarity better? arXiv:2203.04946 [cs].

[Kwolek, 2005] Kwolek, B. (2005). Face Detection Using Convolutional Neural Networks and Gabor Filters. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Duch, W., Kacprzyk, J., Oja, E., and Zadrożny, S., editors, *Artificial Neural Networks: Biological Inspirations – ICANN 2005*, volume 3696, pages 551–556. Springer Berlin Heidelberg, Berlin, Heidelberg. Series Title: Lecture Notes in Computer Science.

[Laparra et al., 2016] Laparra, V., Ballé, J., Berardino, A., and Simoncelli, E. P. (2016). Perceptual image quality assessment using a normalized Laplacian pyramid. *Electronic Imaging*, 28(16):1–6.

[Laparra et al., 2010] Laparra, V., Muñoz-Marí, J., and Malo, J. (2010). Divisive normalization image quality metric revisited. *Journal of the Optical Society of America A*, 27(4):852.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. Conference Name: Proceedings of the IEEE.

[LeCun et al., 2012] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012). Efficient BackProp. In Montavon, G., Orr, G. B., and Müller, K.-R., editors, *Neural Networks: Tricks of the Trade: Second Edition*, Lecture Notes in Computer Science, pages 9–48. Springer, Berlin, Heidelberg.

[Li et al., 2022] Li, Q., Gomez-Villa, A., Bertalmío, M., and Malo, J. (2022). Contrast sensitivity functions in autoencoders. *Journal of Vision*, 22(6):8.

[Lin et al., 2019] Lin, H., Hosu, V., and Saupe, D. (2019). Kadid-10k: A large-scale artificially distorted iqa database. In *2019 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–3. IEEE.

[Liu et al., 2023] Liu, Z., Gan, E., and Tegmark, M. (2023). Seeing is Believing: Brain-Inspired Modular Training for Mechanistic Interpretability. arXiv:2305.08746 [cond-mat, q-bio].

[Luan et al., 2018] Luan, S., Zhang, B., Chen, C., Cao, X., Han, J., and Liu, J. (2018). Gabor Convolutional Networks. *IEEE Transactions on Image Processing*, 27(9):4357–4366. arXiv:1705.01450 [cs].

[Malo and Laparra, 2010] Malo, J. and Laparra, V. (2010). Psychophysically Tuned Divisive Normalization Approximately Factorizes the PDF of Natural Images. *Neural Computation*, 22(12):3179–3206.

[Markram and Sakmann, 1995] Markram, H. and Sakmann, B. (1995). Action potentials propagating back into dendrites triggers changes in efficacy of single-axon synapses between layer V pyramidal neurons. *Society for Neuroscience Abstracts*, 21(1-3):2007–2007. Publisher: EurekaMag.

[Martinez-Garcia et al., 2019] Martinez-Garcia, M., Bertalmío, M., and Malo, J. (2019). In Praise of Artifice Reloaded: Caution With Natural Image Databases in Modeling Vision. *Frontiers in Neuroscience*, 13.

[Martinez-Garcia et al., 2018] Martinez-Garcia, M., Cyriac, P., Batard, T., Bertalmío, M., and Malo, J. (2018). Derivatives and inverse of cascaded linear+nonlinear neural models. *PLOS ONE*, 13(10):e0201326. Publisher: Public Library of Science.

[Mehrotra et al., 1992] Mehrotra, R., Namuduri, K. R., and Ranganathan, N. (1992). Gabor filter-based edge detection. *Pattern Recognition*, 25(12):1479–1494.

[Miller et al., 2022] Miller, M., Chung, S., and Miller, K. D. (2022). Divisive feature normalization improves image recognition performance in alexnet. In *International Conference on Learning Representations*.

[Narayan, 1997] Narayan, S. (1997). The generalized sigmoid activation function: Competitive supervised learning. *Information Sciences*, 99(1):69–82.

[Olah et al., 2017] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature Visualization. *Distill*, 2(11):e7.

[Olshausen and Field, 1996] Olshausen, B. and Field, D. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 281:607–609.

[Ponomarenko et al., 2015] Ponomarenko, N., Jin, L., Ieremeiev, O., Lukin, V., Egiazarian, K., Astola, J., Vozel, B., Chehdi, K., Carli, M., Battisti, F., and Jay Kuo, C.-C. (2015). Image database TID2013: Peculiarities, results and perspectives. *Signal Processing: Image Communication*, 30:57–77.

[Ponomarenko et al., 2009] Ponomarenko, N., Lukin, V., Zelensky, A., Egiazarian, K., Astola, J., Carli, M., and Battisti, F. (2009). TID2008 – A Database for Evaluation of Full- Reference Visual Quality Assessment Metrics.

[Pérez et al., 2020] Pérez, J. C., Alfarra, M., Jeanneret, G., Bibi, A., Thabet, A., Ghanem, B., and Arbeláez, P. (2020). Gabor Layers Enhance Network Robustness. arXiv:1912.05661 [cs].

[Regan, 1991] Regan, D. (1991). *Spatial vision.* Vision and visual dysfunction ; v. 10. Macmillan Press, Basingstoke.

[Ringach, 2002] Ringach, D. (2002). Spatial structure and symmetry of simple-cell receptive fields in macaque primary visual cortex. *J. Neurophysiol.*, 88(1):455-463.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. Number: 6088 Publisher: Nature Publishing Group.

[Schwartz and Simoncelli, 2001] Schwartz, O. and Simoncelli, E. P. (2001). Natural signal statistics and sensory gain control. *Nature Neuroscience*, 4(8):819–825. Number: 8 Publisher: Nature Publishing Group.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs].

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going Deeper with Convolutions. arXiv:1409.4842 [cs].

[Teo and Heeger, 1994] Teo, P. and Heeger, D. (1994). Perceptual image distortion. In *Proceedings of the International Conference on Image Processing*, pages 982–986.

[Veerabadran et al., 2021] Veerabadran, V., Raina, R., and Sa, V. R. d. (2021). Bio-inspired learnable divisive normalization for ANNs.

[Vila-Tomás et al., 2022a] Vila-Tomás, J., Hernández-Camara, P., Li, Q., Hepburn, A., Laparra, V., and Malo, J. (2022a). Basic psychophysics of deep networks trained to reproduce segmentation, maximum differentiation and subjective distortions. In Parraga, A. and Otazu, X., editors, *Workshop on Deep Learning in Vision Science*, Barcelona, Spain. Univ. Barcelona, Iberian Conf. Percept.

[Vila-Tomás et al., 2022b] Vila-Tomás, J., Hernández-Camara, P., and Malo, J. (2022b). A psychophysical turing test for artificial networks devoted to vision. In Santos, R., editor, *Workshop on Evaluating Artificial Intelligence*, Bristol, UK. Bristol Univ. Dept. Eng. Math,.

[Watson and Solomon, 1997] Watson, A. B. and Solomon, J. A. (1997). Model of visual contrast gain control and pattern masking. *J. Opt. Soc. Am. A*, 14(9):2379–2391.

[Zhang et al., 2018] Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. (2018). The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. arXiv:1801.03924 [cs] version: 2.