

**MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS**



VNIVERSITAT  
E VALÈNCIA

**TRABAJO DE FIN DE MÁSTER**

**AUTONOMOUS DRIVING  
SEMANTIC SEGMENTATION WITH  
DIVISIVE NORMALIZATION**

**AUTHOR:**

**PABLO HERNÁNDEZ CÁMARA**

**TUTORS:**

**VALERO LAPARRA PÉREZ-MUELAS**

**JESÚS MALO LÓPEZ**

**MAY 2022**





VNIVERSITAT  
D VALÈNCIA



Escola Tècnica Superior  
d'Enginyeria **ETSE-UV**

## **MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS**

### **TRABAJO DE FIN DE MÁSTER**

# **AUTONOMOUS DRIVING SEMANTIC SEGMENTATION WITH DIVISIVE NORMALIZATION**

**AUTHOR:**

**PABLO HERNÁNDEZ CÁMARA**

**TUTORS:**

**VALERO LAPARRA PÉREZ-MUELAS**

**JESÚS MALO LÓPEZ**

---

**TRIBUNAL:**

PRESIDENTE/A:

VOCAL 1:

VOCAL 2:

**FECHA DE DEFENSA:**

**CALIFICACIÓN:**



# Abstract

One of the key problems in computer vision is adaptation: models are too rigid to follow the variability of the inputs. The canonical computation that explains adaptation in sensory neuroscience is divisive normalization, and it has appealing effects on image manifolds. In this work we show that including divisive normalization in current deep networks makes them more invariant to non-informative changes in the images.

In this work we addressed two standard problems in computer vision as a proof of concept of the benefits of divisive normalization: classification and segmentation. Results show that the inclusion of divisive normalization in the architectures lead to better classification and segmentation results with respect to conventional networks without divisive normalization. Particularly, the segmentation gain increases steadily when dealing with images acquired in bad weather conditions. In addition to the results on the autonomous driving Cityscapes and Foggy Cityscapes datasets, we explain these advantages through visualization of the responses: the equalization induced by the divisive normalization leads to more invariant features to local changes in contrast and illumination<sup>1</sup>.

---

<sup>1</sup>Some of the results of this Master Thesis have been submitted to the 2022 IEEE International Conference in Image Processing (ICIP). Also, it is going to be continued with a PhD called “Aprendizaje Profundo Bio-Inspirado (Bio-Deep)”.

The code is accessible and can be found at: <https://github.com/pablohc97/TFM>



# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                     | <b>9</b>  |
| 1.1      | Motivation . . . . .                    | 9         |
| 1.2      | Objectives . . . . .                    | 12        |
| <b>2</b> | <b>Models</b>                           | <b>13</b> |
| 2.1      | Convolutional neural networks . . . . . | 13        |
| 2.1.1    | Convolution . . . . .                   | 13        |
| 2.1.2    | Pooling . . . . .                       | 14        |
| 2.1.3    | Transposed convolution . . . . .        | 16        |
| 2.2      | Divisive normalization . . . . .        | 16        |
| 2.2.1    | Original formulation . . . . .          | 17        |
| 2.2.2    | Qualitative benefits . . . . .          | 18        |
| 2.2.3    | Current formulation . . . . .           | 20        |
| 2.2.4    | Implementation . . . . .                | 21        |
| 2.2.5    | Implementation tests . . . . .          | 22        |
| 2.3      | Classification models . . . . .         | 24        |
| 2.4      | Segmentation models . . . . .           | 27        |
| 2.4.1    | U-Net . . . . .                         | 27        |

---

|          |                                   |           |
|----------|-----------------------------------|-----------|
| 2.4.2    | Tested models . . . . .           | 28        |
| <b>3</b> | <b>Experiments</b>                | <b>31</b> |
| 3.1      | Computational resources . . . . . | 31        |
| 3.2      | Classification . . . . .          | 31        |
| 3.2.1    | Dataset . . . . .                 | 32        |
| 3.2.2    | Results . . . . .                 | 33        |
| 3.3      | Segmentation . . . . .            | 37        |
| 3.3.1    | Dataset . . . . .                 | 37        |
| 3.3.2    | Metric . . . . .                  | 40        |
| 3.3.3    | Results . . . . .                 | 41        |
| 3.3.4    | Discussion . . . . .              | 46        |
| <b>4</b> | <b>Conclusions</b>                | <b>49</b> |
| <b>5</b> | <b>Extensions</b>                 | <b>51</b> |
| <b>6</b> | <b>Bibliography</b>               | <b>53</b> |
|          | <b>List of Figures</b>            | <b>59</b> |
|          | <b>List of Tables</b>             | <b>61</b> |



# Chapter 1

## Introduction

### 1.1 Motivation

An autonomous driving vehicle, also known as self-driving car (although they do not have to be just cars), are vehicles that are capable of sensing their environment and moving safely with little or no human input. They are one of the most important improvements of technology. Thanks to them, humans will be able to travel or transport goods in a more efficient way, faster, cheaper and with less accidents. According to the Society of Automotive Engineers (SAE International), there are six levels of automation according to the amount of human intervention and attentiveness that the car requires [1]. This classification goes from level 0, which means that there is no automation, to level 5, in which no human intervention is required at all. Although it is true that many advances have been made since the first autonomous driving cars appeared at the end of the 20<sup>th</sup> century [2], currently the most advance self-driving cars are between levels 2 and 3 in the automation scale.

In order to achieve their complex task, autonomous vehicles use many sensors to be able to measure what is happening around them, to sense their environment, in order to take the correct action according to it. These sensors includes global positioning systems (GPS), radio detection and ranging (RADAR), laser imaging detection and ranging (LIDAR) or ultrasonic sensors [3]. However, the most used and important sensors are image sensors, cameras, which record the environment of the car. Processing this information with computer vision and with the information of the other sensors, the car can take the correct decisions.

To perform the recognition using the images from the cameras, some image processing and computer vision tasks have to be done. Computer vision has different branches such as image classification, with or without localization; object detection; instance segmentation or semantic segmentation, as can be seen in Figure 1.1.

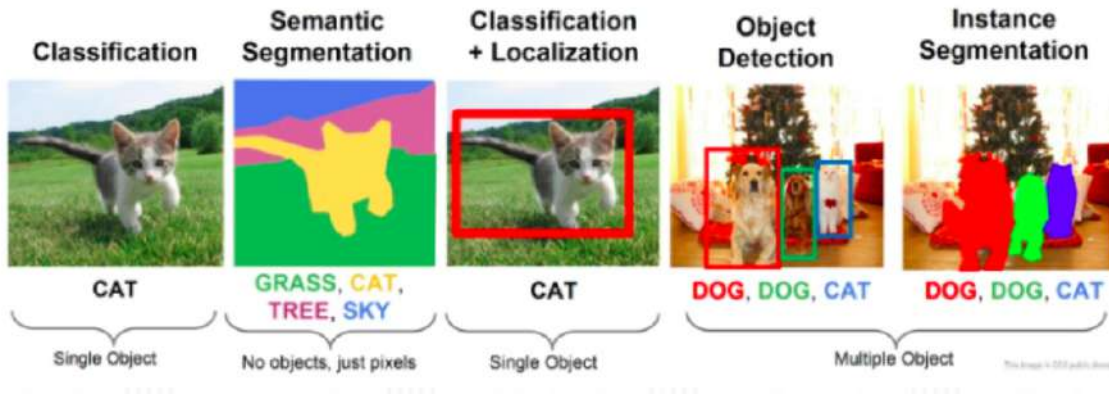


Figure 1.1: Computer vision branches example. On one hand, classification and localization try to identify and locate respectively in the image one single object. Object detection is the extension to more than one object per image. On the other hand, segmentation tries to associate each pixel of the image with a label. Depending on how it processes the same category it can be semantic segmentation (same category has the same label) or instance segmentation (each individual has a label, even if they are of the same category). [4]

Classification is the most fundamental problem because it assumes that there is only one object per image and the output of the model is a discrete label, which corresponds with the object of the image. Also, it can include localization, where in addition to the discrete label the model predicts a bounding box around the object boundary but it also assumes that there is only one object per image. Object detection is the next step and it is the generalization of image classification and localization to the case when there is more than one object per image. Segmentation is completely different. In this case each pixel of the image is associated with a label. In the case of instance segmentation, individuals of the same category are distinguished and so the model classifies each instance separately. Nevertheless, probably the most used in autonomous driving is semantic segmentation. It describes the process of associating each pixel of an image with a class label, such as person, road or car, but without differentiating instances of the same class. This process is extremely important for autonomous driving because it allows vehicles to know what is around them and so taking the correct decision. Moreover, it is not only used in autonomous vehicles but also it can be used in multiple applications, for instance in medical diagnosis [5] or flood detection [6]. Although in recent years great advances have been made in this field achieving high accuracy, some of the last fatal accidents that actual autonomous cars have suffered are due to a bad classification of their environment [7]. For this reason, to achieve more secure autonomous vehicles, improvements in semantic segmentation should be done.

A fundamental problem that image models face comes from a poor adaptation to the variability of the input depending on acquisition conditions [8]. For example, texture and color of a single object can be different in different parts of the image, which has a negative impact on the different applications. In this case, segmentation models should be able to achieve good results although the images suffer from these

type of variations. Ideally the models should be able to adapt themselves to this non-informative variations, i.e. they have to be independent of these type of changes. It means that they have to be able to transform the inputs and work with the images in a space which has to be independent of these characteristics. Left part of figure 1.2 shows an example of changes along the image due to non-informative factors. In this case, texture and color of the forest change due to different factors, such as illumination, shadows or atmospheric scattering. While there are some advances on adaptation for instance in color adaptation [9], more advances on analyzing different sources of variability have to be done.

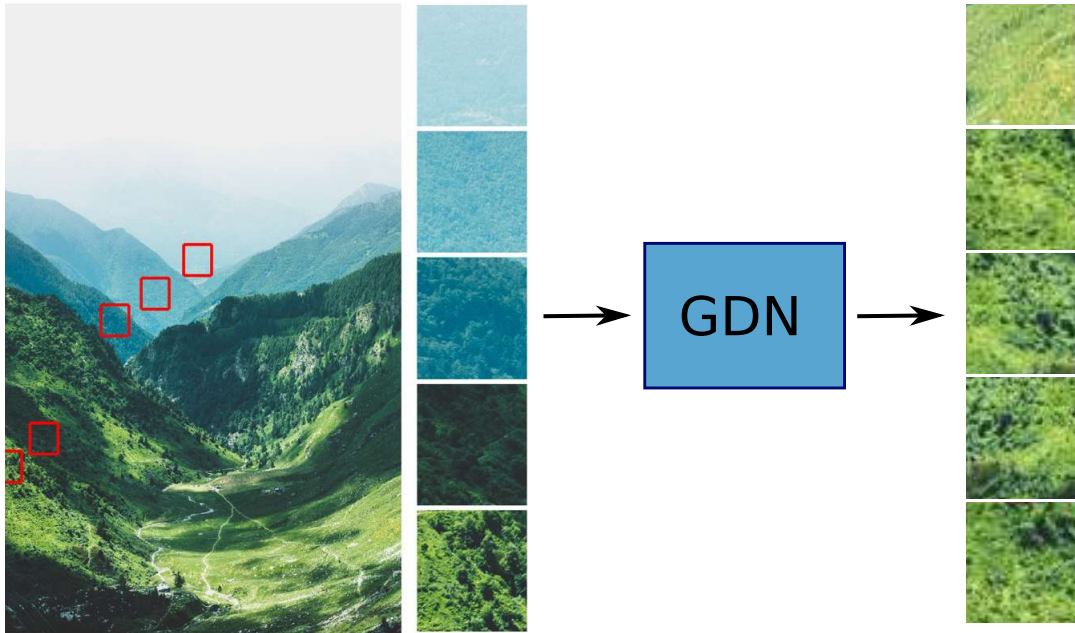


Figure 1.2: Problem that models must face: Texture and color of the forest change due to non-informative factors: illumination, shadows... (highlighted regions in red correspond to the same object: forest). This creates an obvious problem to statistical machine learning models. However, the use of the generalized divisive normalization (GDN) is expected to equalize these differences and thus, with its implementation in the models we expected them to be independent of changes in non-informative characteristics, such as texture or color.

Following this idea of adaptation and the previous computations and experiments [10, 11] a new transformation was introduced in the early 90's as a way of modeling the nonlinear properties of cortical neurons. As it is known, humans are able to recognize a figure even if it is rotated, with different illumination, texture, color or closer or further away. The main idea was to simulate this process that our brain performs almost perfectly. This transformation was called divisive normalization and it is a form of local gain control, in which responses are divided by the pooled activities of their rectified neighbors [12]. After some years of use, this transformation became a standard method for modeling sensory neurons [13] and currently it is canonical computation that explains adaptation in sensory neuroscience.

The use of this transformation, which tries to replicate what is in the human brain, in image problems could be expected to provide a better result than current models that do not implement it. This is because, as shown in the right part of figure 1.2, its use is expected to equalize the images. This has even more relevance in situations that suffer from the problems exposed before, such as low contrast zones or images corrupted with optical scattering. So, a higher improvement is expected in images with these characteristics, like night, shaded or foggy images. With this improvement, it is expected that some accidents such as the one that occurred in 2019, when a Tesla car collided with a white truck, at the moment in which the sun was facing it, killing its occupant [14], can be avoided.

## 1.2 Objectives

The main objective of this project is to improve the semantic segmentation that autonomous vehicles perform, specially in bad weather conditions. It can be seen as an adaptation of the models to changes in color, textures or other non-informative characteristics. To achieve this, we will introduce the use of the neuroscience canonical transformation (divisive normalization) in the models. The result of the objective will be measured performing comparisons between different neural networks models, some of them with divisive normalization and other without it.

The project is divided into two problems: a classification and a segmentation problems. On one hand, the classification problem can be seen as the previous test before facing the real segmentation problem. Here, using the Cifar-10 dataset [15], the models that include the normalization are expected to achieve a better performance. In this problem, three different models with different number of normalization layers are trained and tested with variations of the original Cifar-10 dataset.

On the other hand, the segmentation problem is the main objective of this Master Thesis. In this case the dataset used is the Cityscapes dataset [16], which is made of segmented images taken from a car. To face this problem, three models with different number of normalization layers are trained. All the models of this problem follow the U-Net architecture [17]. In this case, the models are trained with the original dataset and then tested with variations of the dataset which include bad weather conditions, where we expect the higher improvements and the models that include divisive normalization to be more robust.

# Chapter 2

## Models

As stated in section 1.2, two different problems will be faced: classification and segmentation. Each problem has its own models because they have to predict different outputs. However, the models of the both problems had something in common: as the input of the models are images, they use convolutions. Before explaining in detail each architecture of the tested models both for classification and segmentation, it is important to understand what convolutions are and how a convolutional neural network operates.

### 2.1 Convolutional neural networks

Convolutional Neural Networks (CNN's) are a type of artificial neural networks that were introduced in 1980 [18] following the discover of Hubel and Wiesel about the neurons of the visual cortex of a cat [19]. Since then, they began to be widely used [20] and currently they have been applied not only to image problems but also to recommender systems [21], natural language processing [22] or time series analysis [23]. However, CNN's principal use is to face image problems, as in this project, where they have become the state-of-the-art [24, 25]. Basically it is due to the fact that, inspired on the human visual system, they learn patterns inside the images and they are invariant under translations of the input data.

#### 2.1.1 Convolution

Logically, CNN are made up basically with convolutions, which extract the features of the input data, images in this case. Mathematically, a convolution is an operation between two functions that expresses how the shape of one is modified by the other. It is defined as:

$$(f * g)(t) := \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau$$

Where  $f$  and  $g$  are two functions. In the case of image processing, a discrete convolution is defined as a dot product between the input tensor and the filter or kernel tensor. However, most convolutional neural networks and machine learning libraries are actually using the cross-correlation instead of the convolutional operation. It does not change the results because the only difference is if the kernel is flipped. So, in practice, the kernel is projected consecutively over the input tensor and a sum of the element by element multiplication is performed as an output. The application of this discrete operation can be seen in the figure 2.1.

This operation have three main parameters:

- Kernel size: Number of pixels of the kernel or filter. For simplicity they use to be square and use to have an odd number of pixels.
- Stride: Number of pixels that the kernel is moved over the input tensor in each direction after perform each multiplications and sum.
- Padding: It defines how to face with the borders of the image. The usual option is to do a zero padding. It means adding zeros outside the input tensor so that the shapes of the input and output are the same. Other options can be padding with the nearest pixel or the mean of the neighbours. If the padding is not perform, the output shape will be smaller than the input shape.

It is important to highlight that the input tensor do not have to have only one channel, in fact it can have any number of channels. The output of the convolution is known as activation map. As explained above, in case of padding, the initial height and width will be the same as the output ones. Also, it is usual to have more than one filter, so that the final output shape commonly is (*output\_height, output\_width, number\_of\_filters*). Finally, as it is know a non-linear activation function is usually used in order to solve non-linear problems. The most famous and currently used is the Rectified Linear Unit (ReLU), defined as:

$$RELU(x) = \max(0, x)$$

## 2.1.2 Pooling

In addition to the convolutional layers, CNN's usually have pooling layers, which are able to aggregate the context while discarding the spatial information, reducing

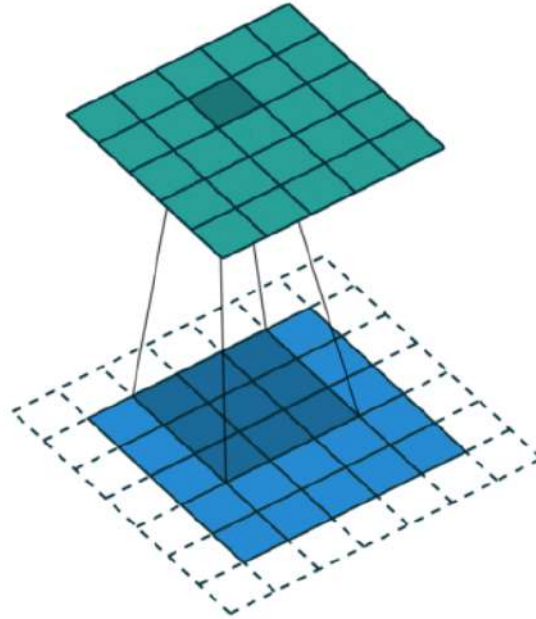


Figure 2.1: Example of a convolution between two tensors, a 3 x 3 kernel over a 5 x 5 input tensor using padding and unit strides. Light blue: Input tensor. Dark blue: Filter or kernel. White: Padding. Green: Output tensor. [26]

the dimensions of the data. Thanks to these layers, the models are able to be invariant to translations in the input data. Pooling layers have only one parameter, the size, as for simplicity they use to be square. There are two types, max pooling gets the maximum value of each sub-matrix of the defined size and average pooling gets the average value of each sub-matrix of sizes:  $(pooling\_size, pooling\_size)$ . It is performed to all the channels of the input data. Its application can be seen in figure 2.2.

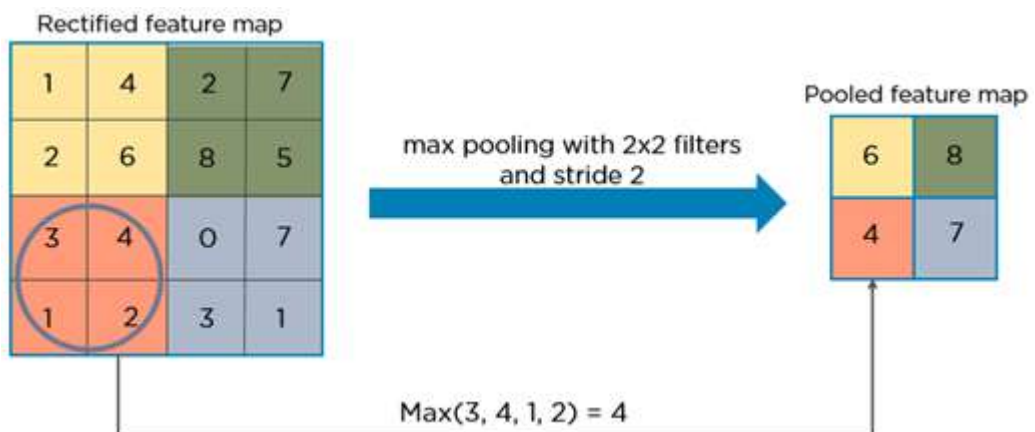


Figure 2.2: Example of a (2, 2) max pooling to a input tensor of shape (4, 4, 1). [27]

### 2.1.3 Transposed convolution

With the previous two layers it is possible to build the majority of convolutional neural networks for classification. However, segmentation architectures and other models that need to up-sample optimally uses one extra layer, know as transposed convolution or deconvolution. It is a layer that perform up sampling of an input tensor, from a low resolution to a higher resolution. At high level, it can be seen as the opposite process of the usual convolution and pooling layer. As opponnent to the convolution, which performs a many-to-one relationship, the deconvolution is a one-to-many relationship. Here, each value of the input tensor is multiply by the kernel and this tensor is located in the correct position to sum up with the expected shape. The final step is to sum over all the sub-tensors, as can be seen in the figure 2.3, where an example of the use of this layer is shown.

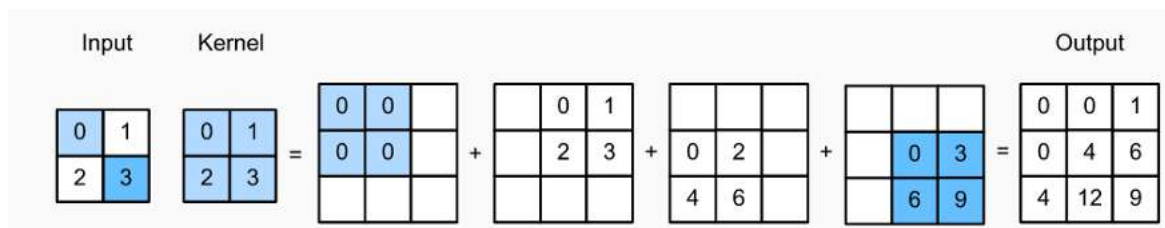


Figure 2.3: Example of a deconvolution from an input of shape (2,2) to an output of (3,3) using a kernel of size (2,2), unit stride and no padding. [28]

## 2.2 Divisive normalization

Currently, the more efficient neural networks are made by a combination of equivariant layers (certain changes in the input produce equivalent changes in the output) and invariant layers (input changes do not affect the outputs). The most exploited example are, as stated previously, the convolutional neural networks (CNN). They have translation equivariance [20] and therefore solve the problem of adaptation under the non-informative change of translations. Pooling layers, always present in CNN's, are the main resource to make the model approximately invariant to small translations. This linear + non-linear structure of conventional neural networks [29] comes from the seminal computation proposed for the brain sensory neurons:

$$\mathbf{x} \xrightarrow{\mathcal{L}} \mathbf{z} \xrightarrow{\mathcal{N}} \mathbf{y}$$

where  $z$  is the intermediate (linear) response, given by  $\mathbf{z} = \mathcal{L} \cdot \mathbf{x}$ . Matrix  $\mathcal{L}$  used to contain the called receptive fields and the original versions of  $\mathcal{N}(\cdot)$  were te inspirations for current sigmoids/ReLU ativation functions in deep-learning [29].



After the explosion of CNN translation invariance, this idea has been extended to other types of symmetries, such as rotations [30] or size scale, but it is complicated to implement this in practise. For instance, it is not trivial to know the order of the layers that produce the equivariances and invariances. As example, capsule neural networks are designed with structures (capsules) to reuse the output from many of this capsules to get a more stable output. With this neural networks some experiments have been tried to understand which layers produce the invariances [31].

In sensory neuroscience, the divisive normalization is the canonical transformation of the non-linear  $\mathcal{N}(\cdot)$ . Following this idea, in this project we will use the divisive normalization to try to implement invariances (adaptative models) to non-informative changes in, for example, contrast.

### 2.2.1 Original formulation

Divisive normalization is a transformation introduced in the early 90's, based on the properties observed in the early stages of the primary visual cortex (area V1) [12]. It has been shown to improve recognition performance when it is applied to deep neural networks [32] and currently it is thought to be in all the visual system, not only in the primary, and also in other brain regions [33]. It has become a standard transformation for describing the visual system neurons [13]. More specifically, it is a form of gain control where lineal responses of neurons are divided by a factor that is the pooled activity of rectified neighbors with some weights. The normalization equation defines this transformation:

$$y_{ip} = \frac{z_{ip}}{\left(\beta_i + \sum_{j p'} \gamma_{ijpp'} |z_{jp'}|^{\alpha_j}\right)^{\epsilon_i}} \quad (2.1)$$

where  $\theta = \{\alpha, \epsilon, \beta, \gamma\}$  are the parameters that are typically fit, although they can be obtained from psychophysical experiments too. The linear response of a neuron ( $\mathbf{z} = \mathcal{L} \cdot \mathbf{x}$ ) tuned to the  $i$ -th feature and to the  $p$ -th spatial location is normalized by the pooled activity of neighbor neurons. The relation between  $\beta_i$  and  $\gamma_{ijpp'}$  determines the level in which the the pool generates the effective inhibition (if  $\beta_i/\gamma_{ijpp'} \gg 1$  it is almost linear and if  $\beta_i/\gamma_{ijpp'} \ll 1$  it is highly non-linear). The interaction kernel in the denominator,  $\gamma$ , can have whatever dense structure [34]. However, in visual neuroscience the spatial interaction is usually assumed to be convolutional [35].

Divisive normalization appearance in 1992 [12] and its first studies [36] came from a physiological point of view where it was intended to describe brain neurons. In this early stage, this transformation took into account interactions in the most general way possible, both spatial and between channels. However, its optimization was restricted until the use of automatic differentiation in the late 2010s.

After some years of use, it started to be studied from a psychophysical point [37] and it started to be applied to image textures [37], color problems [38, 39] and as a way to improve the image distance metrics [40]. Later it was approached from a statistical point of view and applied to an always important topic, redundancy reduction [41]. Since then it has not stopped being used, for example in classification problems [42], and widely applied to topics such as image compression [43] or denoising [44].

A few years later, automatic differentiation was applied to this transformation [45, 46], which led to more general optimization. However, this caused that unlike before, the linear transformation that is applied previously to normalization does not have a clear physical meaning and therefore the meaning of the  $\gamma_{ijpp'}$  parameter, which collect the interactions, is not fully understood. In fact, the first implementation that performed automatic differentiation [45] did not include spacial interactions, only keeping the interactions between the channels. However, from this point the transformation started to be called generalized divisive normalization (GDN).

Despite this, the advantage of the automatic differentiation has lead to an improvement in a large number of direct applications of this transformation. Advances have been made in the developing of new image similarity metrics [47] and its optimization [48], compression algorithms [49], prior knowledge applied to images [46] or rendering [48].

In recent years it has been possible to develop again the extension in terms of the interactions of the transformation. This “new” transformation, includes not only a general optimization thought the automatic differentiation, but also it recovers the spatial interaction and, therefore, the general interactions. Currently, it is being studied again from a physiological point of view [50]. In addition, it is being used to explore how when used with convolutional neural networks it is similar to our brain’s visual system [50, 51] and how it increases the accuracy of the convolutional classification models that include it [52].

### 2.2.2 Qualitative benefits

However, why using this biological transform to improve artificial networks devoted to, for example, image segmentation? In addition to all the examples of effective use mentioned above, figure 2.4 shows how divisive normalization can be used to solve the problems highlighted in figure 1.2. It considers one of the forest highlighted region in figure 1.2 and shows how the use of divisive normalization transforms the features in a positive way to identify the two regions of the same class (more similar manifolds). The local divisive normalization compensates for the contrast and texture variation along the image due to atmospheric conditions. As a result, while the distribution of the samples from the original luminance channel display separate clusters corresponding to spatial variations of the texture, the different samples have been compacted due to the contrast equalization effect. In this case, the divisive normalization transformation

(with non-trained psychophysical tuned parameter [53]) is applied to the output of the linear part, which in this case is a wavelet transformation [34, 41]. As can be deduced, the  $\gamma$  parameter (the circles in the feature images), which controls the neighborhood that defines the pooling region has a important effect: the division by the local pool moderates the response in regions where contrast is high, in orange, while increases the response in regions of low contrast, in white. So that, we get that this transformation maximizes contrast locally [34], i.e. where the activity of the locality is small, its value will increase and vice versa.

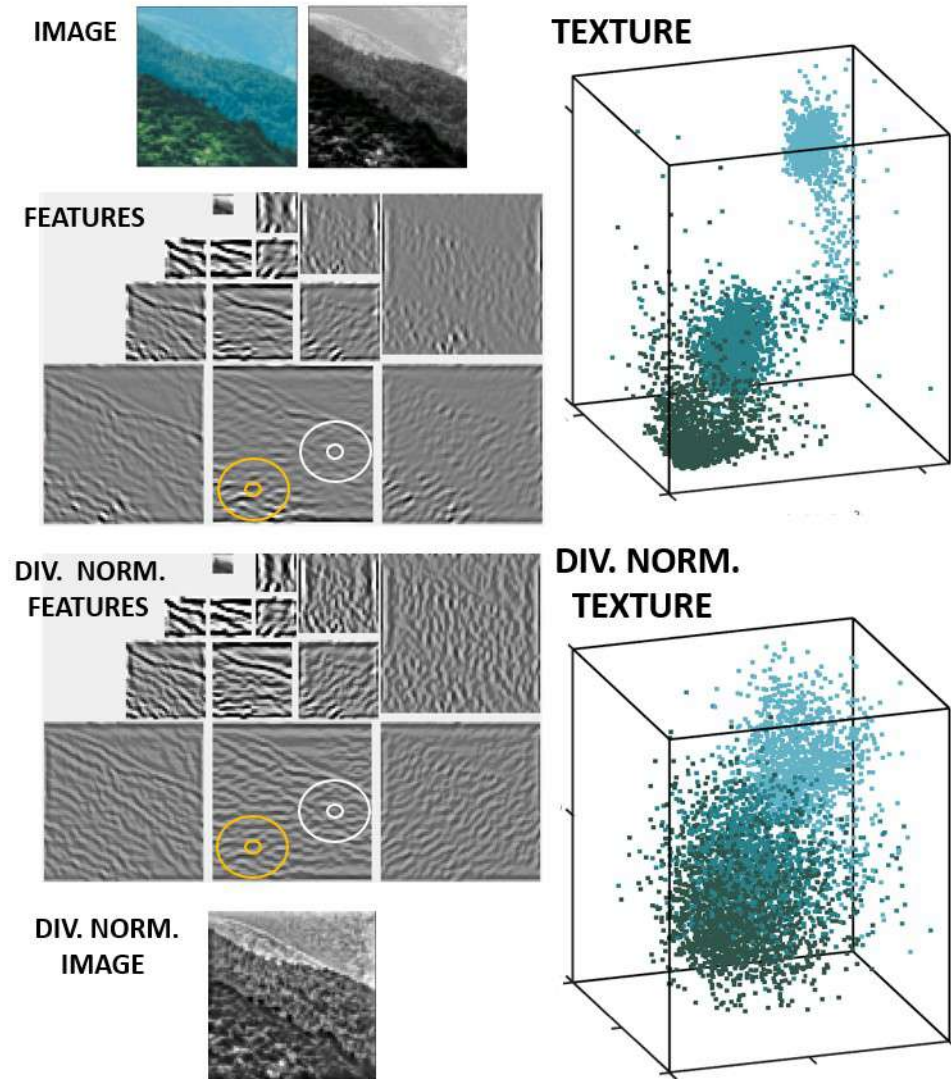


Figure 2.4: At top left we have the luminance input to a linear (wavelet) + non-linear (divisive normalization) model of the V1 brain cortex [53, 34, 41]. Below we show the wavelet linear features ( $\mathbf{z} = \mathcal{L} \cdot \mathbf{x}$ ), the divisive normalization features ( $\mathbf{y} = \mathcal{N}(\mathbf{z})$ ) and the reconstructed image. The scatter plots display the distributions of the image patches before and after the model transformations, from the input and reconstructed images ( $\mathbf{x}' = \mathcal{L}^{-1} \cdot \mathbf{z}$ ). However, it is important to highlight that our brain does **not** perform the image reconstruction.

### 2.2.3 Current formulation

One of the most critical points is the nature of the interaction kernel, the  $\gamma$  parameter (see equation 2.1). Regarding its nature, the example of figure 2.4 shows that include spatial neighborhoods is important to get local contrast equalization to solve the problem introduced by texture variations, for example due to shadows, scattering or fog. Many different approaches have been made to the structure of the interaction kernel: some do not consider spatial interactions (either in a dense [49, 54, 45] or convolutional [55] combination of features); while others take into account spatial interactions but with some restrictions (for example with uniform weights [56], a ring of locations [51, 52] or special symmetries in the space [50]).

In this Master Thesis we used the generalized version of the divisive normalization. Following the intuition pointed out in the previous figures, it means that it is going to include not only a dense interaction between the channels but also convolutional spatial interactions, taking into account the spatial correlation between the neighbors of each pixel of the image. Although the name is the same as the normalization developed in the mid-2010, it is important to highlight that we include both channel and spatial interactions.

The use of this transformation has exhibited to have many advantages. Firstly, it can be used to enhance details and contrast images that are corrupted with optical scattering, like fog, and to boosts the contrast of low-contrast features without introducing significant artifacts [48]. This is highly important because local luminance subtraction an contrast normalization are some of the ways of reducing redundancy in natural images. Secondly, it has shown to be consistent with human perception, showing a correlation with human quality ratings and generating more natural samples of image patches [47, 57, 58, 54]. Thirdly, when used as quality metric it presents a better account of human perceptual judgements than the last published gain-control metrics based on oriented filters [47]. Finally, when it is used in an image compression system it presents a better rate-distorsion performance that the standard JPEG and a extremely improvement in the visual quality of the images [46, 49].

Another important point is that the transformation is well-defined, it means that it is differentiable, continuous and invertible [45]. First, considering the invertibility, taking the partial derivative of 2.1:

$$\frac{\partial y_{ip}}{\partial z_k} = \frac{\delta_{ik}}{\left(\beta_i + \sum_j \gamma_{ijp} |z_{jp}|^{\alpha_j}\right)^{\epsilon_i}} - \frac{\alpha_k \gamma_{ikp} \epsilon_i z_i |z_{kp}|^{\alpha_k - 1} \text{sgn}(z_{kp})}{\left(\beta_i + \sum_j \gamma_{ijp} |z_{jp}|^{\alpha_j}\right)^{\epsilon_i + 1}} \quad (2.2)$$

To ensure the continuity of the function that defines the transformation, all the partial derivatives must be finite for all  $\mathbf{z} \in R^N$ . More specifically, from equation 2.2 the requirement is that all the exponents and the denominator of the divisive normalization must be non-negatives.

It can be proved that the general condition to ensure the invertibility of the divisive normalization is the following [59]:

Let  $D_z$  and  $D_\beta$  be diagonal matrices with the absolute value of the elements of  $z$  and  $\beta$  in the diagonal and  $V$  and  $\Lambda$  the eigenvector and eigenvalue matrix decomposition of  $D_z \cdot \gamma$ . To ensure the invertibility, the matrix  $(I - D_z \cdot \gamma)$  has to be invertible, which leads to  $|I - D_z \cdot \gamma| \neq 0$  and can be expressed as:

$$\text{Eigenvalues}(D_z \cdot \gamma) \leq 1$$

However, the analytical inverse is not computationally efficient because it needs to calculate the inverse of large matrices, which is a slow process. Nevertheless, the transformation can be efficiently inverted using a recurrent process as follows [45]:

$$z_i^{(0)} = \text{sgn}(y_i) (\gamma_{ii}^{\epsilon_i} |y_i|)^{\frac{1}{1 - \alpha_{ii} \epsilon_i}}$$

$$z_i^{(n+1)} = \left( \beta_i + \sum_j \gamma_{ij} |z_j^{(n)}|^{\alpha_{ij}} \right)^{\epsilon_i} y_i$$

## 2.2.4 Implementation

In this Master Thesis, we implemented the generalized divisive normalization layer from equation 2.1 in an automatic differentiation context, in this case using the *Tensorflow* library<sup>1</sup> of Python. Therefore, one may include our layer at any place of any architecture and optimize the network for the desired task. If one want to use our developed GDN layer, it is located in the following GitHub: <https://github.com/pablohc97/TFM>

This layer takes an input tensor of shape (*weight, height, channels*) and returns an output tensor of the same shape. Its unique meta-parameter is the size of the the  $\gamma$  parameter, called *filter\_shape*, which are the squared kernels of the convolutions that the layer performs in the denominator. As explained in the previous section, this  $\gamma$  parameter is where spatial interactions are included. Its default value is (3, 3), so with this value the convolutions will be performed with kernels of shape (3, 3). The first version used convolutions which were performed with a value of 1 for the stride and *zero* padding in order to get an output of the same shape of the input. The second version also used a value of 1 for the stride but used a reflection of the border pixels to perform the padding.

We initialized the layer in two different ways. On one hand, one initialization is that it has no effect in the data, i.e. it is such an identity layer, which only lead the input

<sup>1</sup><https://www.tensorflow.org/>

data as the output of the layer. In this case, as can be easily deduced from equation 2.1, the parameters have to be initialized such as  $\alpha = \beta = \epsilon = 1.0$  and  $\gamma = 0.0$  kernels. On the other hand, the second layer initialization corresponds with a way it smudges the images, mixing the values of the pixels in the kernels. In this case, the initialization of the parameters are  $\alpha = 2.0$ ,  $\beta = 0.001$ ,  $\epsilon = 0.5$  and  $\gamma = 1.0$  kernels.

The parameters have some restrictions in order avoid the denominator could be 0 and therefore avoid an infinite as an output of the layer and to ensure the invertibility of the transformation. They are the following:

- $\alpha$  parameters: They cannot be negative.
- $\beta$  parameters: They cannot be less than  $1^{-15}$  in order to avoid that the whole denominator could be 0.
- $\epsilon$  parameters: They cannot be negative.
- $\gamma$  parameters: They cannot be negative.

One interesting thing about our layer is its number of parameters. As can be deduced from equation 2.1, the number of parameters of the layer is as follow:

- $\alpha$  parameters: Number of Channels.
- $\beta$  parameters: Number of Channels.
- $\epsilon$  parameters: Number of Channels.
- $\gamma$  parameters:  $\text{Filter\_shape} \times \text{Filter\_shape} \times \text{Number of Channels} \times \text{Number of Channels}$

For example, using the default parameter for the filter shape, if the input tensor has a shape  $(32, 32, 3)$  (squared RGB color image of 32 pixels per side) the number of parameters of the layer will be only 90, as can be easily calculated from the before statement.

### 2.2.5 Implementation tests

Before using the developed GDN layer to face the real problems, we carried out several test to check that it works as expected.

## MNIST

Firstly, we tested the identity initialization with the MNIST dataset [60]. This well known dataset has images with shape  $(28, 28, 1)$ , i.e. gray-scale squared images of 28 pixels. For our test, we built a simple model with only a GDN layer and we checked that the layer has 12 parameters:  $\gamma : 3 \times 3 \times 1 = 9$ ,  $\alpha : 1$ ,  $\beta : 1$  and  $\epsilon : 1$ . After verifying that all weights were initialized correctly, we passed the images through the model without performing any training. It means that the weights of the layer were as initiated, they did not changed. In the figure 2.5 we show the result of this test, which, as expected, returned the image without any change.

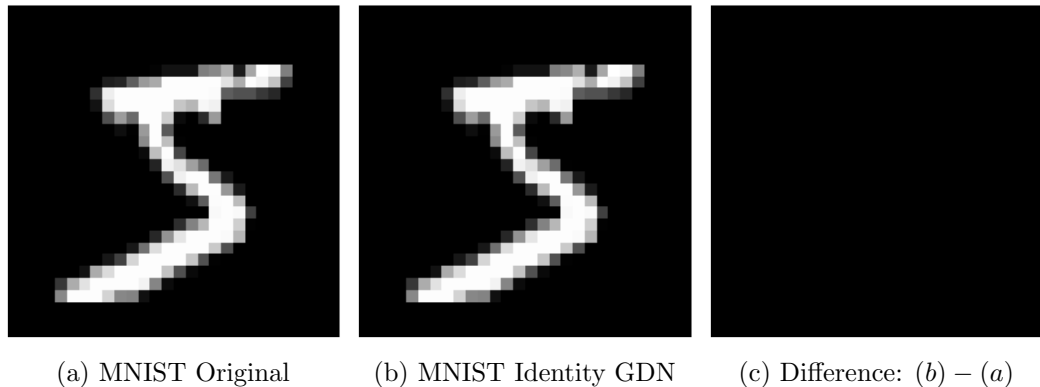


Figure 2.5: Left figure (a) shows the original MNIST image before pass through the model. Center figure (b) shows the same MNIST image after pass through the model, which only apply a GDN layer without training with the identity initialization. As expected, with this initialization the output of the layer is the same as the input. Right figure (c) shows the difference between the two previous images:  $(b) - (a)$ . The three images have the same range between 0 and 1. As expected, the difference is zero everywhere.

Then, we performed the same process with the other weight initialization. In this case, the result was a blurred version of the image as we show in figure 2.6. This is the expected effect due to the initialization of the  $\gamma$  kernels to the value of 1.

Also, in order to be sure that the layer performed what we wanted, we did the same process that the GDN layer performs by hand using the *Scipy* library<sup>2</sup>. In this case we initialized the weights so that they blurred the images. The result of this experiment is that the same pixel values were obtained using the GDN layer and the *Scipy* library. In figure 2.7 we these two outputs and the difference between them, which, as expected, it is zero everywhere.

---

<sup>2</sup><https://scipy.org/>

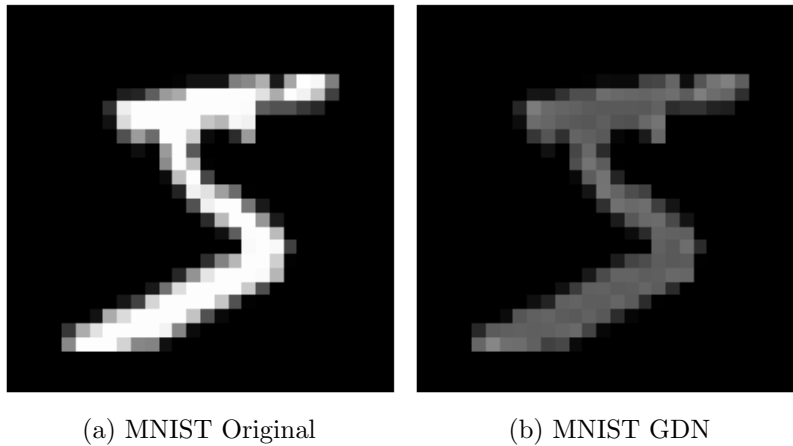


Figure 2.6: Left image (a) shows an original MNIST image before pass through the model. Right image (b) shows the same MNIST image after pass through the model, which only apply a GDN layer without training with the blurred initialization. As expected, the initialization of the  $\gamma$  parameters to 1 smudge the image.

### Fashion MNIST

We did also a test with the Fashion MNIST dataset [61], which is also made-up by grey-scale images of  $28 \times 28$  pixels. As before, we defined a simple sequential model with only one GDN layer and we checked that it had 12 parameters (if the number of channels of the input images do not change, the number of parameters of the layer have to be the same). After verifying that the weights were correctly initialized, we passed the images through the model without performing any training, with the weights as were initiated. In figure 2.8 we show how the output of this model create the effect of blurring the images.

## 2.3 Classification models

With all the previous layers exposed and developed we built the classification models. For this problem, which in fact is the final test before facing the main segmentation problem, we tested three different models. In all the models we included three convolution layers with  $(3, 3)$  kernels. The number of filters of each one of the layers were 8, 16 and 32. They all included RELU as activation function, unit strides and same padding with 0 to fed the borders. After each convolutional layer we included a  $(2, 2)$  average pooling layer. They reduced the shape of the images to a half of their input values. Finally, after flatten layer we used a final 10 neurons dense layer with a soft-max activation function. This function, which is a generalization of the logistic regression, is defined as:



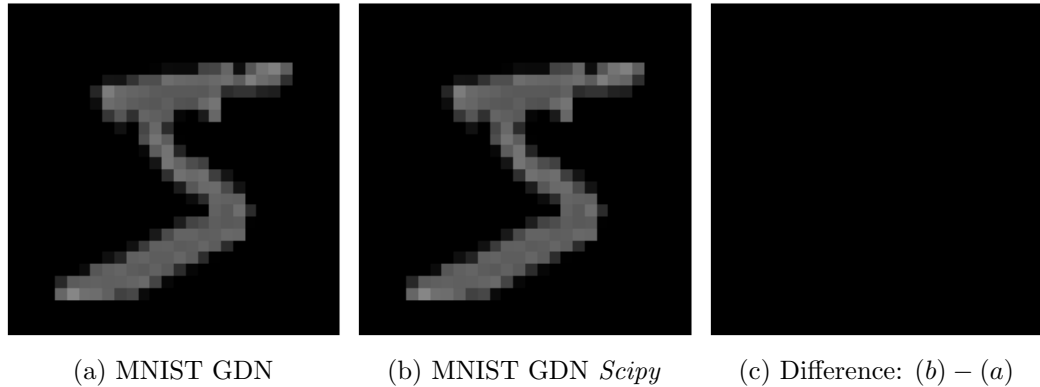


Figure 2.7: Left image (a) shows the output of a MNIST image after pass through the model which apply a GDN layer without training with the blurred initialization. Center image (b) shows the same MNIST image performing the same operations than the GDN layer with *Scipy*. Right image (c) shows the difference between the two previous images:  $(b) - (a)$ . The three images have the same range between 0 and 1 and, as expected, the difference is zero everywhere.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad for \quad j = 1, 2, \dots, K \quad (2.3)$$

Where  $K$  is the number of classes. It is used for normalize the output of a network to a probability distribution over the predicted output classes. It means that the output of this function is the probability of each image of belonging to each class.

The difference between the three models was the number of GDN layers they had. Regarding where to include the divisive normalization layers, it is important to recall the color and texture problems highlighted in figure 1.2. The usual brain and color appearance models [38, 62] state that color normalization may be addressed by a single normalization of the photo-receptors (just before the first convolution), while texture equalization over space may require normalization to deeper stages (after convolutions) where filters tuned to specific patterns have emerged. Following this idea, we created three models: the simpler one, with no GDN layers as a base-line. In the second one we included only one GDN layer, located just before the first convolution, in order to try to solve the color normalization. The last model we created included not only this first GDN layer but also two more GDN layers at deeper stages, before each one of the three convolution layers. This model was supposed to be able to face not only the color but also the texture normalization. We named the models as No-GDN, 1-GDN and 3-GDN respectively. In figure 2.9 we show the architecture of the model with three GDN layers. As can be deduced from the previous explanation, the other two models are simpler versions of this one, with only the first or any of the GDN layers (in green).

For the 1-GDN we initialized the GDN layers as the identity and we used (7, 7) kernels for the  $\gamma$  parameter for each one of the three GDN layers and we fixed the

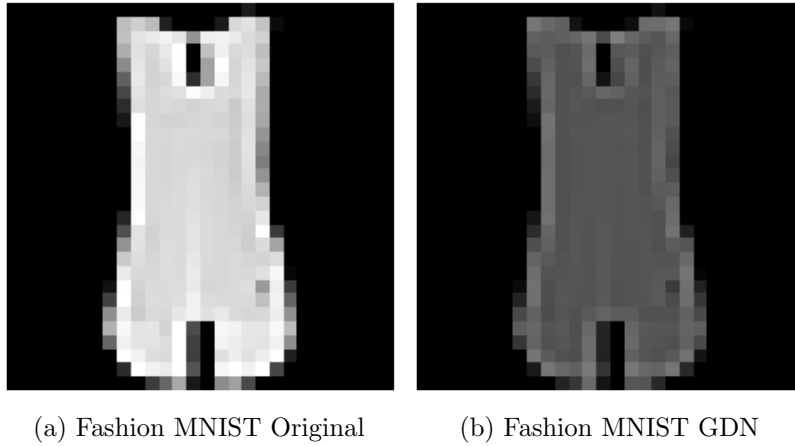


Figure 2.8: Left image (a) shows an original Fashion MNIST image. Right image (b) shows the same Fashion MNIST image after pass through a GDN layer model with the blurred weights. The initialization of the  $\gamma$  parameters 1 smudges the image.

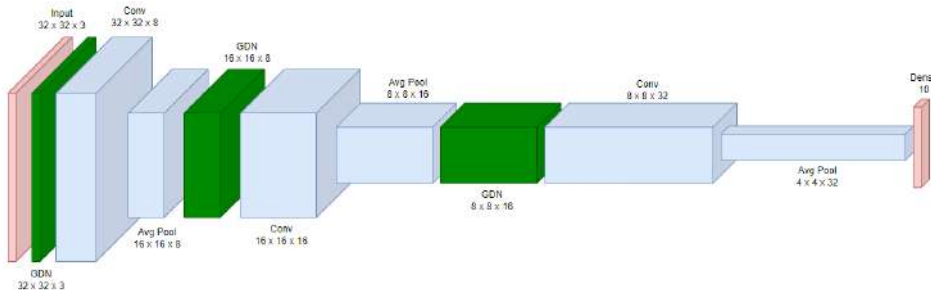


Figure 2.9: Architecture of the classification problem model with three GDN layers (in green). The model with only one GDN layer had only the first green layer. The model without GDN layers did not have any of the green layers.

exponents to be non-trainable, which we found to be more stable, so that  $\alpha = \epsilon = 1$ . For the 3-GDN model, it was also initialized as identify with non-trainable exponents and with a  $(7, 7)$ ,  $(5, 5)$  and  $(3, 3)$  GDN layer kernels.

The models had 11162, 11606 and 15534 trainable parameters for the No-GDN, 1-GDN and 3-GDN models respectively. The increment from the No-GDN to the 1-GDN was less than a 4% in the number of parameters while the increment from the No-GDN to the 3-GDN was around than a 40%.

## 2.4 Segmentation models

### 2.4.1 U-Net

The U-Net network used a new architecture and it was published in 2015 by the department of computer science of the University of Freiburg [17]. This network shown some advantages such as it can be trained fast with few images but achieve good results. It was first developed for biomedical image segmentation, where it won the ISBI cell tracking challenge 2015<sup>3</sup> by a large margin, but after its success it has been extended to many other fields.

The architecture of the U-Net networks can be divided in two parts as can be seen in the figure 2.10. On one hand, there is the contracting path, where the input images pass, as usual, through different convolutions and max pooling layers. These layers are arranged in blocks, formed by two 3 x 3 convolution layers with RELU as activation function and a final 2 x 2 max pooling layer which reduces the shape of the image to a half before pass them to the next block. Each block has the same number of filters for each one of the two convolutions and the number of filters in the convolutions usually increase with the blocks in powers of two, it is doubled. In this part, the convolutions extract the existing characteristics in the images, although when the max pooling layers reduce the dimensions the spatial information is lost.

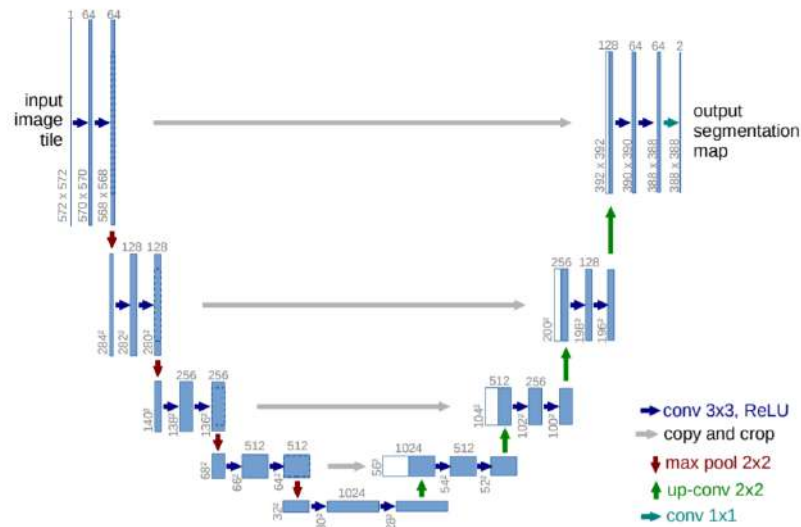


Figure 2.10: Architecture of the original U-Net network. Input images have a shape of 572 x 572 and they are reduced to 32 x 32 in the bottleneck. Then, with 2 x 2 deconvolutions they are up-sample to the original shape and the pixel label is predicted (only two classes). Each blue box represents a feature map, with the number of channels on the top and the shape at the left. The different layers are highlighted with the different colorful arrows. [17]

<sup>3</sup><https://biomedicalimaging.org/2015/program/isbi-challenges/>

At the end of the contracting path, the images reach the point where they have the smallest desired size. This point is sometimes called bottleneck. It is formed by two more  $3 \times 3$  convolutions layers with RELU. It has the maximum number of filters in the convolutions and here the images have the smallest size.

After the bottleneck, next part is the expansive path. Here, images come with their smallest shape from the bottleneck and through different blocks of up-convolutions the images are resized to their original shape. As in the contracting part, each one of these expansive blocks is formed by one up-sampling  $2 \times 2$  up-convolution (instead of a max pooling, because now it needs to up-sample the images), a concatenation with the corresponding feature map of the contracting path and two  $3 \times 3$  convolution layers with RELU. In this case, images go from a small size up to a bigger size and so, as opposite of the contracting path, each block has now less convolution filters than the previous one. This expansive part does the opposite of the contracting, here the spatial information is recovered by gradually up sampling the images to get the precise location of the characteristics extracted by the contracting path. The final step is a  $1 \times 1$  convolution to convert the 64 channels of the last expansive block to the number of classes to predict, two in this case.

Finally, what makes the U-Net different from other previous architectures: the concatenations between the contracting and expansive paths, also known as skip unions. These unions, that perform a concatenation, pass the feature maps from each one of the contracting path blocks to the appropriate expansive path block, so that the shapes are the same. They help to get more precise segmentation decisions because they are a way to restore the spatial information lost through the contracting part. This network has a symmetric architecture with the shape of the letter U, where does its name come from, U-Net.

## 2.4.2 Tested models

To face our segmentation problem, we modified the original U-Net architecture to include our GDN layer in the contracting path. Following this idea and similarly to the classification problem, we built three different models. The only difference in the models was the number of GDN layers. Regarding where to include the GDN layers, we followed the same idea as in the classification models. Hence, we built a simple model based on the U-Net architecture without any GDN layer as a base-line. Our second model included only one GDN layer which should take care of the color normalization and therefore it was located before the first convolution. The last model included four GDN layers, the first one and three more in deeper stages of the contracting path in order to try to solve the texture normalization. We named the models as No-GDN, 1-GDN and 4-GDN respectively. In figure 2.11 we show the architecture of the model which includes 4 GDN layers. As the U-Net, they were formed by two main different parts, the contracting and the expansive paths.



Figure 2.11: U-Net architecture model built for segmentation including 4 GDN layers (in green). Black arrows represent the skip unions. The model with only one GDN layer had only the first green layer. The model without GDN layers did not have any of the green layers. The consideration of the GDN layers only increases the number of parameters by an 1.8% in the case of the 4 GDN layers.

Our three models are prepared to work with  $96 \times 256$  color images, so that the inputs of the models have a shape of  $(96, 256, 3)$ . The contracting path included four convolution blocks. Each one included two convolutions followed by a batch normalization and a RELU activation layer. The parameters of the convolutions were  $(3, 3)$  kernels, *zero* padding and unit strides, so that the convolution did not change the shape of the images. The number of filters that each convolution had was the same for the two convolution layers inside each block, but it changed between the different blocks. The first block had 16 filters for the convolutions and in each block the number of filters is doubled. Following this rule, the last block had 128 filters. After each one of the convolution blocks, a max pooling layer was included. The use of these pooling layers reduced the shape of the image by a half. It means that the images arrived to the bottleneck with a shape of  $(6, 16, 128)$  because they had been reduced by a half four times. The use of this contracting path allowed the models to extract the features of the images.

In the bottleneck we included one extra convolution block with other two convolutions, as in the contracting path. In this case the two convolutions had 256 filters. The difference was that in this case there was not a max pooling layer, so it did not reduce the shape of the images, which arrived to the expansive path with a shape of  $(6, 16, 256)$ .

The expansive path was almost symmetric to the contracting path. It included four convolution blocks identical to the blocks of the contracting path. The difference was that instead of a max pooling layer after the blocks, it included transposed convolutions before the blocks. These transposed convolutions doubled the shape of the images each time because its parameters were  $(3, 3)$  kernels, zero padding and  $(2, 2)$  strides. In this part, the number of filters of the transposed convolutions was reduced by a half each time, from 256 to 32. After each transposed convolution but before each convolution block, a concatenation was included. This skip unions concatenated the output of each transposed convolution with the output of the convolution blocks of the contracting path. They were very helpful to restore the spatial information lost in the contracting path by passing the corresponding feature maps of the contracting path to the expansive

path. After the concatenations, the convolutions blocks were included. They had the same parameters than in the contracting path and as opposite the number of filters of the convolutions was reduced by a half each time instead of doubled. So that, the number of filters changed from 128 to 16 in this case. So that, taking into account the number of filter of the convolution blocks and how the transposed convolution reduce the shape of the images, the output images of the expansive path had a shape of (96, 256, 16). Then, we included a final convolution with a (1, 1) kernel to map each pixel to the probability of belonging to each class, so that this layer had 30 filters and a soft-max activation function.

As stated before, the difference between the models was only the number of GDN layers. The first model had four GDN layers which were located before each one of the convolution block of the contracting path. The second model had only one GDN layer, located before the first convolution block, so that it was the first layer of the model and it received the input images. Finally, the last model had no GDN layers and therefore it was exactly as described above. For the GDN layers we initialized them as the identity and we used (3, 3) kernels for the  $\gamma$  parameter and fixed the exponents to be non-trainable, which we found to be more stable, so that  $\alpha = \epsilon = 1$ .

It is important to highlight that with the introduction of GDN layers the symmetry of the U-Net architecture was broken. To maintain it, inverse GDN layers should be introduced in the expansive path. However, as in this case the aim of the model is not to restore the images but to predict the class of each pixel (segment the images) they are not needed.

The models had 2749902, 2749986 and 2798482 trainable parameters for the No-GDN, 1-GDN and 4-GDN models respectively. The increment from the No-GDN to the 1-GDN was around a 0.003% in the number of parameters while the increment from the No-GDN to the 3-GDN was less than a 2%.

# Chapter 3

## Experiments

### 3.1 Computational resources

We used three different resources to build the scripts and train the models. They were a laptop, the Google Colab resource and the Iris server from the Department of Electronic Engineering of the University of Valencia.

More particularly, the laptop model with which we did most of the tests was a MSI GF63. This laptop had an Intel i7 processor of tenth generation, a ram of 16 GB and a NVidia GeForce GTX 1650 GPU. However, we changed the laptop at the middle of the project and currently it is a HP Omen 15 with an AMD Ryzen 7 processor, a ram of 16 GB and a NVidia GeForce RTX 3070 GPU. About the Google Colab resource, we used it always with the GPU accelerator environment in order do the first test of training the models. Finally, to train the models we used the Iris server. It is a server from the Department of Electronic Engineering of the University of Valencia. More specifically, it has two AMD Vega 10 Radeon Instinct MI25 GPU's which have been used to train the different models in a faster way than with the the laptop or Google Colab.

### 3.2 Classification

The first problem we faced was a classification problem. As stated before, it can be seen as the final test before the segmentation problem. Here, as well as in the segmentation, an improvement is expected with the models which include GDN layers.

### 3.2.1 Dataset

We used the Cifar-10 dataset<sup>1</sup> for the classification problem. This well-known dataset is simple because it only has ten classes or labels to predict. It has 60000 RGB color images of shape  $(32, 32, 3)$ , divided into train (50000 images) and test (10000 images). First, we extracted 5000 random images from the training dataset to create a validation dataset. Then, the number of images was as follow: 45000 images for train, 5000 images for validation and 10000 images for test.

We decided to use this dataset because as stated before, it is well-known, easy because it only has ten classes and has small images. However, it also implied a problem: the dataset is too perfect. The images are very artificial, with the object to classify always located in the centre of the image and with always the same illumination, without any shadow or contrast changes. With these perfect images, divisive normalisation role is small. For this reason, we decided to modify the dataset to include local contrast and luminance changes, so that we could check if the divisive normalization take this into account and led the models to a higher accuracy.

We performed two changes. On one hand, we changed the luminance of the images darkening them. To do this, we randomly selected a *limit* value in the range  $[2, 4]$  and then we generated 32 equally spaced values in the range  $[-limit, limit]$ . Then, we calculated the response of a sigmoid function to these values so that we got 32 values in the range  $(0, 1)$ . After that, we randomly selected an orientation (horizontal or vertical) and then a direction (from left to right or from right to left for the horizontal orientation and from top to bottom or from bottom to top for the vertical orientation). Then, we multiplied the rows or columns (depending on the selected orientation) of the image times the 32 generated values in the selected direction. The result of this transformation is a local darkening performed randomly in any direction by rows or by columns.



Figure 3.1: Left image shows an original  $32 \times 32$  RGB image from the Cifar10 dataset. Center image shows the same image after changing its luminance locally in the vertical direction, going from dark in the top to lighter in the bottom. Right image shows the same image after changing its contrast locally in the horizontal direction, going from more contrast in the left to less contrast in the right. We performed the two transformations with the explained process.

<sup>1</sup><https://www.cs.toronto.edu/~kriz/cifar.html>



On the other hand, we did a similar transformation to change the contrast of the images. The process was the same than for the luminance transformation, but when we had the 32 (0,1) values and the orientation and direction selected we did not performed a simple multiplication. In this case, we used the ImageEnhance function from the PIL library<sup>2</sup> to enhance the contrast. This function takes a factor to perform the enhancement and we used the 32 generated values to enhance the contrast in the randomly selected orientation and direction. In figure 3.1 we show an original image and its two modifications performed following the above explanation.

With these two modification we build the dataset to train the models. This dataset has one third of the original images, one third of images changed with the explained contrast transformation and one third changed with explained luminance transformation. We selected randomly the transformation applied to each image (identity, luminance or contrast).

To sum up, we have the following datasets:

- Original dataset:
  - Train images: 45000 original
  - validation images: 5000 original
  - Test images: 10000 original
- Modified dataset dataset:
  - Train images: 15000 original + 15000 luminance + 15000 contrast
  - validation images: 1666 original + 1667 luminance + 1667 contrast
  - Test images: 3334 original + 3333 luminance + 3333 contrast

### 3.2.2 Results

The metric we wanted to maximize in this problem is the accuracy because we were classifying the images. The accuracy is defined as percentage of images correctly labeled by the model. We trained the models exposed in section 2.3 with the modified dataset created through the transformations exposed before. Moreover we did a small data augmentation with a random horizontal flip and a random horizontal or vertical shift of a maximum of the 15% image in order to scroll object to classify from the center of the image. We made different tests to select the initial learning rate independently (finally all of them fitted to the same value,  $10^{-3}$ ) and we reduced it as the training went on through a callback. We also did different test to get the correct patience (100 epochs) and factor (0.5 of the previous learning rate) to reduce the learning rate through a callback. We used a batch size of 32 and Adam for the optimizer. We trained each model ten times with different seeds during 2000 epochs. To sum up, we used the following hyper-parameters to train the models:

---

<sup>2</sup><https://pillow.readthedocs.io/en/stable/>

- Training process parameters:
  - Number of seeds: 10
  - Loss function: Categorical cross-entropy
  - Metric: Accuracy
  - Optimizer: Adam
  - Initial learning rate:  $10^{-3}$
  - Batch size: 32
  - Epochs: 2000
  - Learning rate callback patience: 100
  - Learning rate callback factor: 0.5

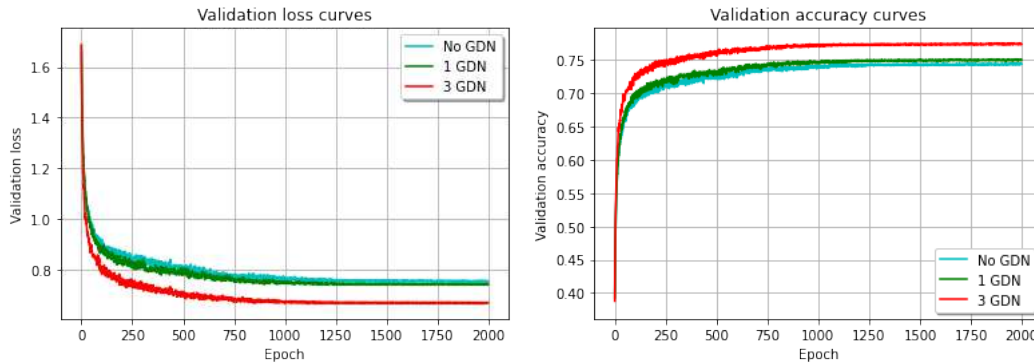


Figure 3.2: Mean validation loss (left image) and mean validation accuracy (right image) curves for the No-GDN, 1-GDN and 3-GDN models. Each curve shows includes a line with the mean values and a plotted zone which corresponds with the mean standard errors.

In figure 3.2 we plot the mean validation accuracy and loss of the training process. As it is possible to see, the 3 GDN model has less loss and higher accuracy. However, due to the large scale of the graphs it is not possible to properly appreciate the differences between the No-GDN and 1-GDN curves and the mean standard errors. For these reasons, we decided to change the visualisation.

In figure 3.3, for each one of the ten training processes (with different seeds), we calculated the validation loss and validation accuracy difference between the 3-GDN model and the No-GDN model and between the 1-GDN model and the No-GDN model. Then, we calculated and plotted the mean and mean standard errors of these differences. From these plots we see that for the validation loss the numbers are negative and the curve of the difference between 3 GDN - No GDN is under the curve of the difference between 1 GDN - No GDN. It implies that the 3-GDN model has less error than 1-GDN and No-GDN and the 1-GDN has less error than the No-GDN but more than the 3-GDN. The validation accuracy curve shows the same effect but with the accuracy. It has positive numbers and the 3 GDN - No GDN curve is on the top of

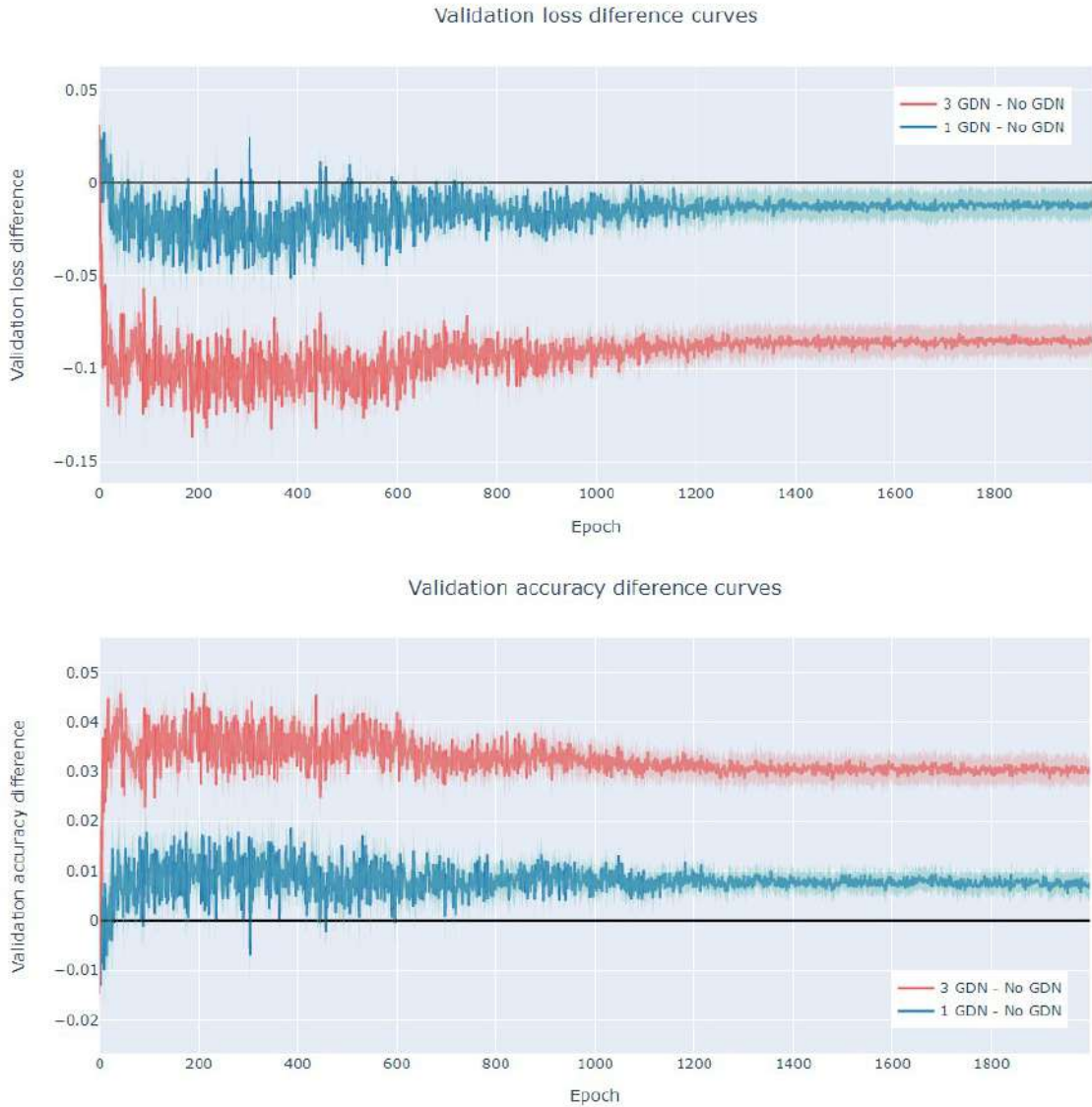


Figure 3.3: Mean differences in the validation loss (top image) and validation accuracy (bottom image) curves between the 1 and 3 GDN models compared with the No-GDN model. Each curve shows the mean difference (red and blue lines) and the mean standard errors (red and blue color zones).

the 1 GDN - No GDN curve, which implies that the 1 GDN model has more accuracy than the No-GDN model and the 3-GDN model has more accuracy than both.

However, these results obtained from the training curves were on the validation dataset. To test the models we used the test dataset. During the training we saved the weights with higher accuracy in the validation dataset and we restored them and used the models with these weights to test both in the modified and the original test datasets. To confirm our hypothesis we performed a two related T-test to find out if it was true that the values obtained were actually different. In this test the null hypothesis was that the two independent samples had identical average (expected)

values. We used a p-value of 0.05 as the limit to reject the null hypothesis.

| Mean accuracy and standard deviation |                 |                        |                        |
|--------------------------------------|-----------------|------------------------|------------------------|
| Dataset                              | No GDN          | 1 GDN                  | 3 GDN                  |
| Original                             | $0.75 \pm 0.01$ | $0.76 \pm 0.01$ (1.3%) | $0.77 \pm 0.01$ (2.7%) |
| Modified                             | $0.74 \pm 0.01$ | $0.75 \pm 0.01$ (1.4%) | $0.77 \pm 0.01$ (4.1%) |

| T test and p-value |                         |   |
|--------------------|-------------------------|---|
| Dataset            | No GDN - 1 GDN          | No GDN - 3 GDN                          |
| Original           | T = -1.8, p-val = 0.107 | T = -5.3, p-val = $4.80 \times 10^{-4}$ |
| Modified           | T = -2.1, p-val = 0.070 | T = -6.9, p-val = $7.20 \times 10^{-5}$ |

Table 3.1: Top panel shows the mean and standard deviation accuracy performance in test. We show the improvements of the use of GDN layers with regard no using GDN in parenthesis. The bottom panel shows the results of the T-test and the p-values when comparing the use of GDN layers with regard no using DN.

The top panel in table 3.1 shows the mean and standard deviation test results for the two datasets: the original and the modified. Also, it shows the mean improvements of the use of GDN layers with regard no using GDN layers. The bottom panel shows the results of the T-test performed between the ten accuracy values obtained by the models that use GDN layers with regard the models without GDN layers.

From this table we can get different conclusions. On one hand, the use of one or three GDN layers gives an increase in accuracy in the two datasets. In addition, using three GDN layers the accuracy obtained is always higher than the obtained with just one GDN layer. This is consistent with the theory because as expected the first GDN should perform a color normalization (which we did not modified). On the other hand, as expected, with the modified dataset we get smaller accuracy because the images have more variability. It happens in the No-GDN and the 1-GDN models but not for the 3-GDN models, which shows that with 3-GDN layers the models are able to adapt themselves to these changes and not perform a reduction in the accuracy. Also it is important to highlight that the increase in accuracy when comparing the No-GDN models with regard the 3-GDN models is higher for the modified dataset. Finally, from the T-test and its p-values we obtain that the null hypothesis is always rejected (p-value smaller than the limit we fixed before: 0.05) when comparing the no GDN model with the 3 GDN model for the two dataset conditions, which means that the difference in performance found is significant.

## 3.3 Segmentation

After facing the classification problem, our next step was to face a segmentation problem. Here, as stated in section 1.1, the objective is to predict a label for each one of the pixels in the image.

### 3.3.1 Dataset

To achieve the objectives that have been exposed, we used the Cityscapes Dataset [16]. It is a large-scale dataset created using sequences of stereo video of 1.8 seconds of duration recorded in the streets of 50 cities of Germany as seen in figure 3.4. The videos have been recorded for several months in spring, summer and autumn (not in winter) and always in daytime with good or medium weather conditions. They have been manually selected to contain a large number of dynamic objects, varying scene layout and the background.



Figure 3.4: 50 locations from Germany where the videos and images were recorded.

Each one of these videos of 1.8 seconds have 30 frames and from them, the 20<sup>th</sup> frame is extracted. With these frames, the images are annotated, labeling each pixel with a class, and the dataset is formed. The annotation for each image (also known as mask or ground truth) has 30 different classes of objects, each one represented by a different colour. In table 3.2 is possible to see the different classes and their associated colors of the masks. After a sign in process<sup>3</sup>, the images and masks are available and it is possible to download them.

<sup>3</sup><https://www.cityscapes-dataset.com/>

| Class | Class name    | Colour ( $R, G, B$ ) | Class | Class name    | Colour ( $R, G, B$ ) |
|-------|---------------|----------------------|-------|---------------|----------------------|
| 1     | Unlabeled     | (0, 0, 0)            | 16    | Traffic sign  | (220, 220, 0)        |
| 2     | Dynamic       | (111, 74, 0)         | 17    | Vegetation    | (107, 142, 35)       |
| 3     | Ground        | (81, 0, 81)          | 18    | Terrain       | (152, 251, 152)      |
| 4     | Road          | (128, 64, 128)       | 19    | Sky           | (70, 130, 180)       |
| 5     | Sidewalk      | (244, 35, 232)       | 20    | Person        | (220, 20, 60)        |
| 6     | Parking       | (250, 170, 160)      | 21    | Rider         | (255, 0, 0)          |
| 7     | Rail track    | (230, 150, 140)      | 22    | Car           | (255, 0, 0)          |
| 8     | Building      | (70, 70, 70)         | 23    | Truck         | (0, 0, 70)           |
| 9     | Wall          | (102, 102, 156)      | 24    | Bus           | (0, 60, 100)         |
| 10    | Fence         | (190, 153, 153)      | 25    | Caravan       | (0, 0, 90)           |
| 11    | Guard rail    | (180, 165, 180)      | 26    | Trailer       | (0, 0, 110)          |
| 12    | Bridge        | (150, 100, 100)      | 27    | Train         | (0, 80, 100)         |
| 13    | Tunnel        | (150, 100, 100)      | 28    | Motorcycle    | (0, 0, 230)          |
| 14    | Pole          | (153, 153, 153)      | 29    | Bicycle       | (119, 11, 32)        |
| 15    | Traffic light | (250, 170, 30)       | 30    | License plate | (0, 0, 142)          |

Table 3.2: Names and colors of the 30 different classes in the ground truth images.



Figure 3.5: Example of one of the Cityscapes images (left image) and its mask (right image).

The dataset is divided in two, according to the quality of the annotations. First, there are 5000 images with fine annotations as seen in the figure 3.5. This dataset is also subdivided into train, validation and test formed by 2975, 500 and 1525 images respectively. After download it only the train and validation sets had annotated images but the test set did not have annotated images. For this reason, we used the 500 validation images as the test dataset. Then, we extracted 300 random images from the train dataset to use them as the validation dataset. Finally we randomly shuffled the three datasets because the images were ordered by cities.

The images had an original shape of 1024 x 2048 pixels, which we resized to smaller values in order to improve the training time of the models. As it is possible to see in figure 3.5, images had the front of the car with the Mercedes Benz star at the bottom of the image. As this happens in all the images, we decided to cut the bottom pixels. After the crop, images had a size of 768 x 2048 and we resized them to 96 x 256 to maintain the aspect ratio. We used nearest neighbor for the resize method of the masks

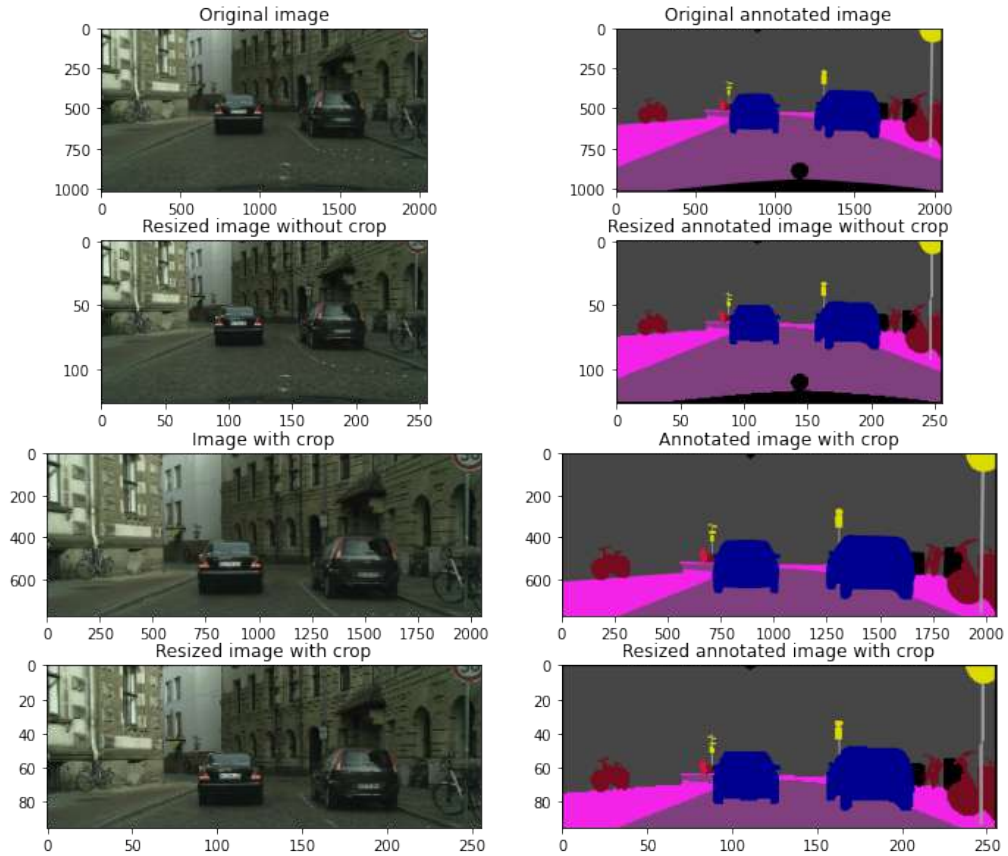


Figure 3.6: The first row shows an original image and its mask without crop and resize. The second row shows the resized image and its mask but without the crop. They maintain the aspect ratio with the original images (first row). The third row shows an image and its mask after the crop but without the resize. Finally, the fourth row shows the resized and cropped image and its mask. They maintain the aspect ratio with the cropped images (third row). The image and its mask without the crop have been resized from 1024 x 2048 to 128 x 256 and the ones with the crop from 768 x 2048 to 96 x 256

to prevent the resize from creating new colours not presents in the original 30 classes. Also, we divided the images, not the masks, by 255 in order to normalize them to the range  $[0, 1]$  because this will prevent large values in the back-propagation algorithm when training the models. In figure 3.6 is possible to see one image and its mask before and after this process.

Also, in addition to the fine annotated images the Cityscapes dataset has 20000 images with coarse polygonal annotations, which have much less details. However, as the annotation masks of these images are not as complete as the previous ones, these images will not be used.

We decided not to modify the images because they were taken in real conditions and therefore they are not as perfect as the Cifar10 images that we used for the classification problem. As they were taken from streets of real cities, they naturally present shad-

ows, different textures and different contrasts, which will allow us to test our divisive normalization without modify the train images.

Additionally to the original Cityscapes dataset, we also used the Foggy Cityscapes dataset [63]. It includes degraded versions of the same scenes simulating poor weather conditions of controlled (low, medium, high) fog severity. Each level is characterized by a constant attenuation coefficient (0.005, 0.1 and 0.2) which correspond with a visibility range of 600, 300 and 150 meters for the low, middle and high fog respectively. This is important to test our models because texture spatial varying degradation is a major problem due to the fog.



Figure 3.7: Example of one of the Cityscapes images and its variation with different levels of fog. Left image shows the original image and the other images show the different fog level images, from low to high (left to right).

To sum up, we have the following datasets:

- Original dataset:
  - Train images: 2675 original
  - validation images: 300 original
  - Test images: 500 original
- Low fog dataset:
  - Test images: 500 low fog
- Middle fog dataset:
  - Test images: 500 middle fog
- High fog dataset:
  - Test images: 500 high fog

### 3.3.2 Metric

Before expose the results it is important to explain which is the metric we are going to focus on. As it is known, the accuracy is just one of the many metrics we can use to check the performance of a model. It is easy to understand, because in segmentation



it is defined as the percent of pixels that are classified correctly. However, it has some problems, for example that it does not take care of the class imbalance and the place where every pixel is located. In fact, in segmentation problems the intersection over union (IoU) metric is more used. This metric, also known as Jaccard index is defined as:

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Intersection}}$$

It is calculated by taking the area of overlap between the predicted segmentation and the ground truth, taken from the masks, divided by the area of union between the predicted segmentation and the ground truth. Figure 3.8 shows a representation of this metric. Its range is  $[0, 1]$ , where 0 means no intersection and 1 indicates a perfect segmentation intersection. In the case of multi-class segmentation, such as in our problem, where there are 30 classes to predict, the IoU is calculated by taking the IoU of each class and then taking the average. In our problem we are going to use this mean IoU as the main metric.

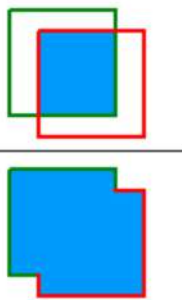
$$IOU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{img}}{\text{img}}$$


Figure 3.8: Visual representation of the intersection over union (IoU) metric [64].

### 3.3.3 Results

First of all, we trained the three different models stated in section 2.4.2 with the original dataset because as stated before they already have enough variability. We made different tests to select the correct loss function. Ideally we should use the metric (mean IoU) as the loss function, but it is not differentiable (like the accuracy) so that we have to use another one. Although we expected it to be the categorical cross-entropy because we are getting the probability of each pixel of belonging to each one of the thirty classes, we got that the curves of the IoU and the categorical cross-entropy were too different. After trying different loss functions, we get that mean absolute error (mae) was the best. Also, we used a batch-size of 64 and Adam optimiser with a learning rate of 0.001, which we reduce as the training went ahead through a callback. We also performed different test to find the more accurate values for the patience (50 epochs) and the factor in which the learning rate is reduced (0.5 of the previous learning rate). With this settings, we trained each model ten times with different seeds

during 500 epochs. We kept the models with higher Intersection over Union (IoU) in validation. To sum up, we used the following hyper-parameters to train the different models:

- Training process parameters:
  - Number of seeds: 10
  - Loss function: Mean absolute error
  - Metric: Intersection over union
  - Optimizer: Adam
  - Initial learning rate:  $10^{-3}$
  - Batch size: 64
  - Epochs: 500
  - Learning rate callback patience: 50
  - Learning rate callback factor: 0.5

In figure 3.9 we show a summary of the training curves. For each one of the ten training processes, we calculated the validation loss, validation accuracy and validation IoU difference between the 4GDN model and the No-GDN model and between the 1 GDN model and the No-GDN model. Then, we calculated and plotted the mean and mean standard errors of these differences.

However, these results obtained from the training curves were only in the validation dataset. To test our models, we loaded the saved weights (models with higher IoU in validation) and used them for test in the four different datasets (the original Cityscapes and the Foggy Cityscapes with its three fog levels). It is important to highlight that all the models were only trained with the original-good weather images and at this point we were testing them in four different datasets (original-good weather and the three versions of poor weather).

To confirm our hypothesis we performed a two related T-test to find out if it was true that the values obtained were actually different. In this test the null hypothesis was that the two independent samples had identical average (expected) values. We used a p-value of 0.05 as the limit to reject the null hypothesis.

The top panel in table 3.3 shows the test mean IoU and standard deviation results in the four weather conditions for the ten models trained only using the original (good weather conditions) images. Also, it shows the mean improvements of the use of GDN layers with regard no using GDN layers for each weather condition. The middle panel shows how the mean test IoU changes in poor conditions with regard to regular weather. Finally, the bottom panel shows the results of the T-test performed between the ten IoU values obtained by the models that use GDN layers with regard the models without GDN layers.

| IoU performance |                 |                         |                         |
|-----------------|-----------------|-------------------------|-------------------------|
| Dataset         | No GDN          | 1 GDN                   | 4 GDN                   |
| Original        | $0.75 \pm 0.02$ | $0.75 \pm 0.01$ (0.0%)  | $0.77 \pm 0.02$ (2.7%)  |
| Low fog         | $0.65 \pm 0.02$ | $0.65 \pm 0.04$ (0.0%)  | $0.70 \pm 0.02$ (7.7%)  |
| Middle fog      | $0.54 \pm 0.03$ | $0.53 \pm 0.04$ (-1.9%) | $0.62 \pm 0.03$ (14.8%) |
| High fog        | $0.40 \pm 0.05$ | $0.38 \pm 0.04$ (-5.0%) | $0.48 \pm 0.03$ (20.0%) |

| IoU reductions (due to fog) |        |        |        |
|-----------------------------|--------|--------|--------|
| Dataset change              | No GDN | 1 GDN  | 4 GDN  |
| Original-low fog            | -13.3% | -13.3% | -9.1%  |
| Original-middle fog         | -28.0% | -29.3% | -19.5% |
| Original-high fog           | -46.7% | -49.3% | -37.7% |

| T test and p-value |                         |   |
|--------------------|-------------------------|---|
| Dataset            | No GDN - 1 GDN          | No GDN - 4 GDN                          |
| Original           | T = -1.2, p-val = 0.254 | T = -3.5, p-val = $6.51 \times 10^{-3}$ |
| Low fog            | T = 0.8, p-val = 0.449  | T = -7.0, p-val = $6.58 \times 10^{-5}$ |
| Middle fog         | T = 1.5, p-val = 0.179  | T = -8.8, p-val = $1.07 \times 10^{-5}$ |
| High fog           | T = 1.4, p-val = 0.183  | T = -5.9, p-val = $2.35 \times 10^{-4}$ |

Table 3.3: Top panel shows the mean IoU and the standard deviation IoU performance in test for each dataset. The improvements of the use of DN layers with regard no using GDN are shown in parenthesis. The middle panel shows the reductions in mean test IoU when comparing each fog level with the original, i.e. comparing the results of each model in the different datasets. Finally, the bottom panel shows the results of the T-test and its p-values of the use of GDN layers with regard no using GDN.

In figure 3.10 we show an illustrative example of the predictions of the models for each weather condition, which are consistent with average IoUs in the tables.

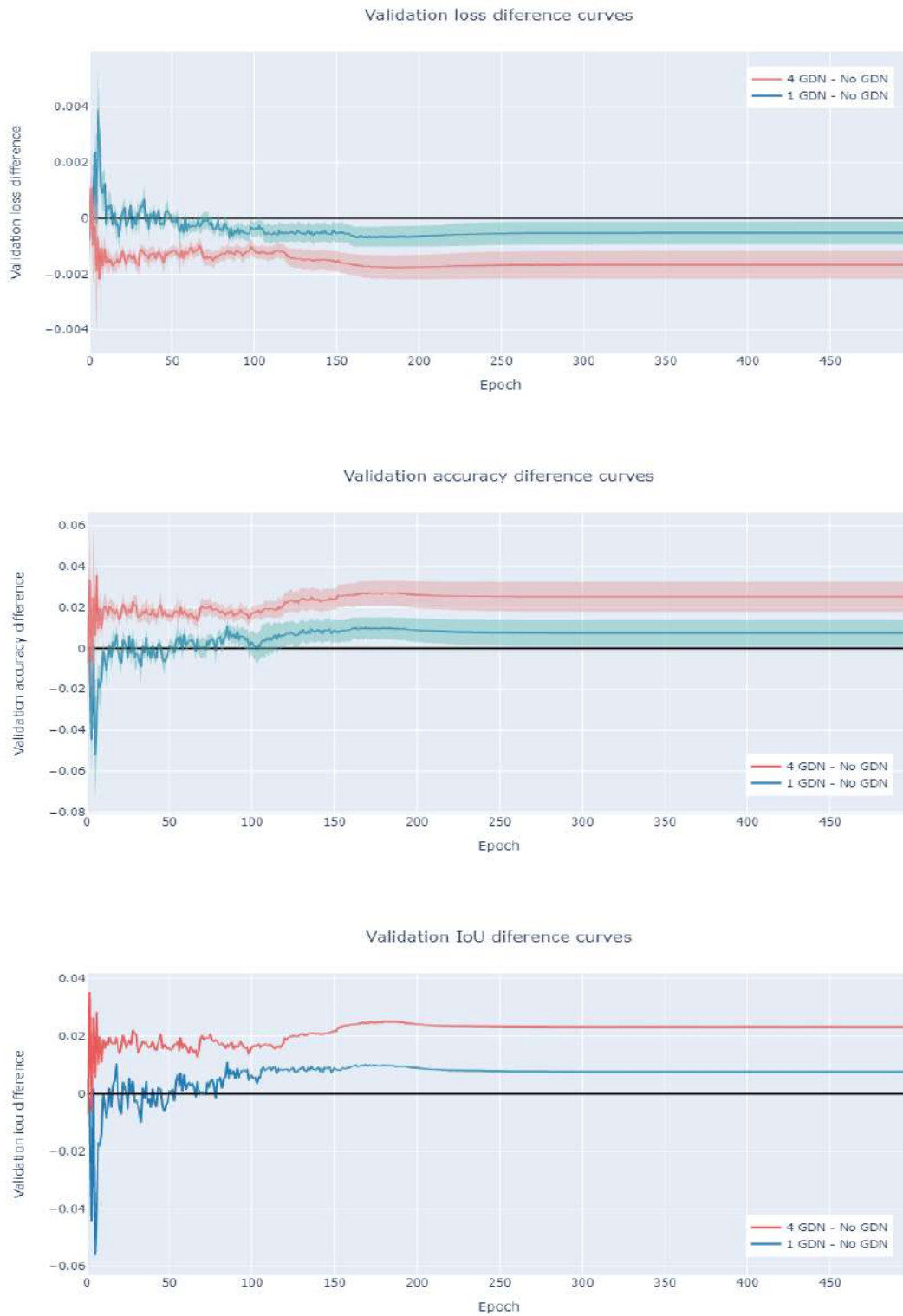


Figure 3.9: Mean differences in the validation loss, accuracy and IoU curves between the 1 and 4 GDN models compared with the No-GDN model. Each curve shows the mean difference (red and blue lines) and the mean standard errors (red and blue color zones).

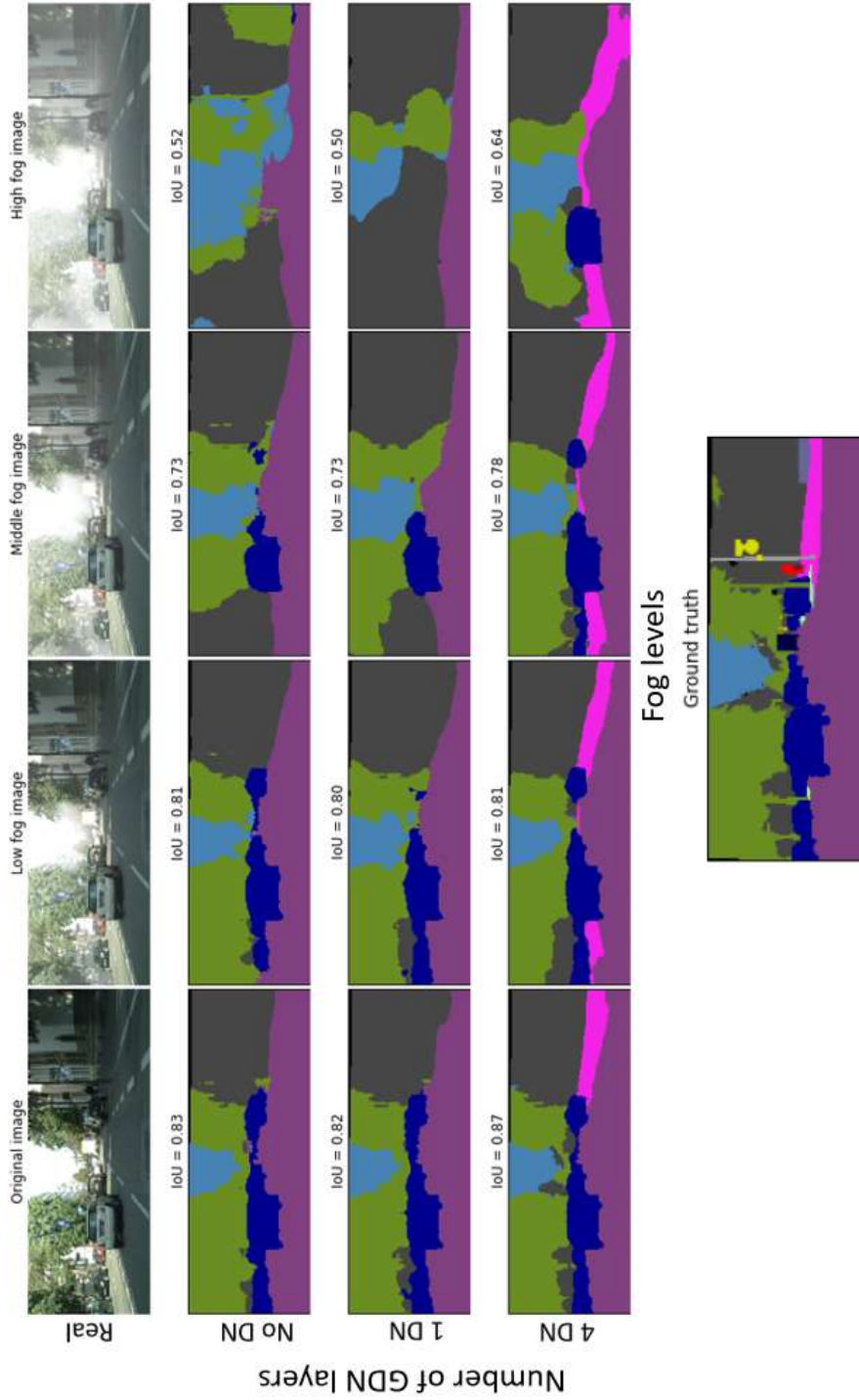


Figure 3.10: Each column shows a different weather condition from no fog (original images in the left column) to high fog images (in the right column). The first row shows the original image that we used to get the predictions. The second, third and fourth rows shows respectively the no-GDN, 1-GDN and 4-GDN model predictions for each weather. Finally, at the bottom the ground truth of the image is shown.

### 3.3.4 Discussion

From the plots in figure 3.9 we see that for the validation loss the numbers are negative and the curve of the difference between 4-No GDN layers is under the curve of the difference between 1-No GDN layers. It implies that the loss of No-GDN model is the higher one, followed by the loss of the 1-GDN model and the 4-GDN model has the smallest error. The validation accuracy curve shows the same effect but in this case it has positive numbers and the 4-No GDN layers curve is on the top of the 1-No GDN layers curve, which implies that the accuracy of the 4-GDN model is higher than the accuracy of the 1-GDN model. The same effect happen with the validation IoU difference curves. From these curve we can conclude that, at least in validation, the higher number of GDN layers the model has, the better results it gets.

The first observation from the table 3.3 is that using 4-GDN layers gives substantial increase in IoU in all cases. Using just 1-GDN seems not to have a really important effect, probably because images do not have much color variation. Second, as expected, for progressively heavier fog, IoU gets reduced in all cases. However the conventional architecture is more sensitive to the decrease in visibility (bigger reductions in performance) than the architecture with 4-GDN layers. And third, the use of 4-GDN always leads to improvements with regard the No-GDN case, but it is important to see that the gains get progressively bigger when the acquisition conditions are poor. From the T-test and its p-values we obtained that the null hypothesis is always rejected (p-value smaller than the limit we fixed before: 0.05) when comparing the no GDN model with the 4 GDN model for all the weather conditions, which means that the results come from independent samples.

From figure 3.10 it is important to realize that accordingly with the table 3.3 results, the IoU gets reduced for progressively heavier fog but the 4-GDN models are less affected by the fog. In fact, only the 4-GDN model is able to get the border in the shadow and preserve the detection of the car in heavy fog.

#### DN effect

Although all the advantages exposed above are due to the use of the divisive normalization, we want to go deeper in its knowledge and we want to know exactly what is its effect though the models. In order to understand it, we explored the effect of the two initial GDN layers of the 4-GDN model.

The first GDN, applied to the input RGB image only seems to boost low values and moderate high values. It is equalizing the pixel values taking care not only of each pixel value but also of their neighbourhood. More interestingly, figure 3.11 shows the effect of the second GDN layer. We selected two of the sixteen feature maps to perform our analysis, one of them tuned to vertical edges (first and third columns) and other tuned to horizontal edges (second and fourth columns), which are key for object

recognition. It is possible to see directly the different features, first and third column figures clearly show vertical white edges while second and fourth column figures show horizontal edges. To illustrate the effect of the divisive normalization we used one image and its version with high fog, which can be seen at the top of figure 3.11. We passed these images through the model and we visualize the horizontal and vertical feature maps just before (first row, named as input) and after (second row, named as output) the second GDN layer.

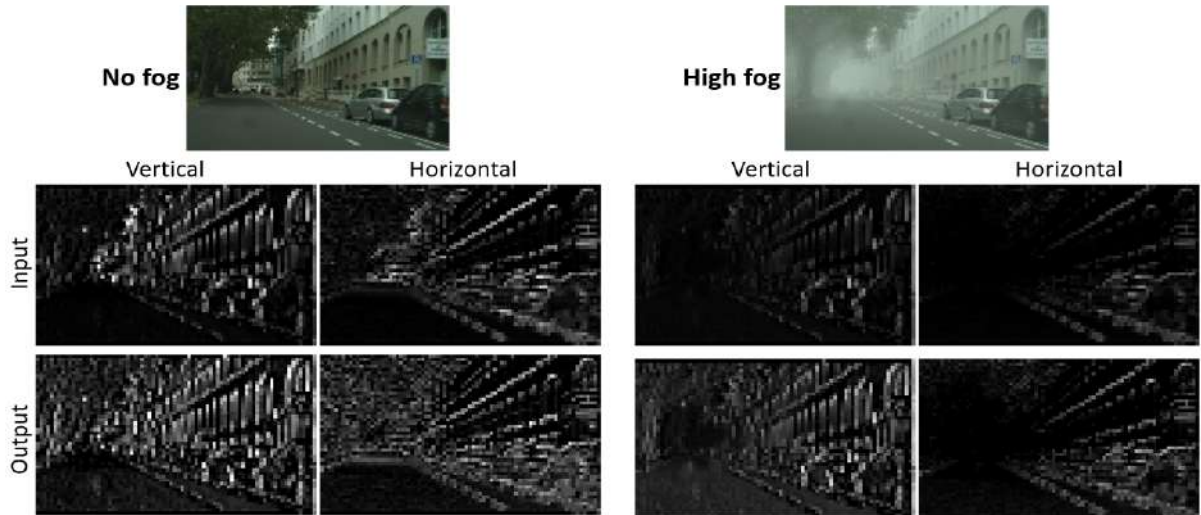


Figure 3.11: Effect of the 2nd-GDN layer in two channels, one tuned to vertical and one tuned to horizontal features. At the top we show the two images (on the left the original and on the right the same high fog image) that we used as input of the model. The first row corresponds with two of the sixteen feature channels just before the second GDN layer of the model. The two columns on the left are the vertical and horizontal features of the original image while the two columns of the right are the same features for the high fog image. Finally, the last row show these two feature maps but after the second GDN layer of the model.

We can obtain different conclusions. First, if we compare the input and output of the no fog image, we see that after the divisive normalization the feature maps show much more details (see the tree at the top left of the vertical feature map or the car at the bottom right of the horizontal feature map) and we can see now deeper part of the image that we almost impossible to distinguish before the divisive normalization (see specially the deepest part of the horizontal feature map).

Secondly, if we compare the inputs of the original image with the inputs of the high fog image we can see the effect of the fog. As it is possible to realize, the input feature maps, both vertical and horizontal, of the high fog image have much less details than the input feature maps of the no fog image. It can be appreciated specially in the deepest part of the image, where the fog is thicker.

Third, we can compare the input and output feature maps of the high fog image. They show that after the divisive normalization the feature maps show much more

details, such as in the no fog scenario. It is possible to appreciate the effect in the horizontal feature map after the divisive normalization, where the upper right edge of the building is more visible and even deeper. However, it is clearly more visible on the vertical feature map, where behind the divisive normalization we can see much better the arches of the building on the right and the tree on the top left.

Finally, if we compare the input and output feature maps of the high fog image with the input feature maps of the no fog image, we can see that the output feature maps of the no fog image look much more like the input feature maps of the no fog image. This is specially that we were looking for: made the feature maps independent of changes in the non-relevant characteristics, such as the changes in the texture introduced by the fog. We have shown that with divisive normalization, intermediate feature maps of the models when they process a fog image look much more similar to the feature maps when they process a no fog image, making the model independent of the fog.



# Chapter 4

## Conclusions

As stated at the beginning of this Master Thesis, the main goal was to improve the semantic segmentation that autonomous vehicles performs, specially in bad weather conditions. To do that, we showed in figure 1.2 that adaptation is one of the key problems that models face. They should be able to transform the images to a space independent of non-informative characteristics, such as color, texture or contrast, to be independent of variations in these features. In figure 2.4 we showed that the canonical neuroscience transformation, the divisive normalization, can solve this problem and we correctly implement its generalized version in an automatic differentiation environment such as Tensorflow. After that, we performed different test, all of them successfully, to test our own implementation of the generalized divisive normalization.

Once we were sure that our implementation did what it was supposed to do, we faced a classification problem as the final test. Here we trained three different models with three, one and any GDN layers with a variation of the Cifar10 database. We obtained promising results. First, the more GDN layers the model had, the higher accuracy it got, both when tested with the original and modified images. Second, as expected the more variability of the modified images led the No-GDN and 1-GDN models to a reduction in the accuracy they obtain. Remarkably, the model with 3-GDN layer was not affected by the more variability of the images, achieving the same accuracy in the both scenarios, which showed that the model was much more flexible and able to work independently of these transformations. Thirdly, as expected, the model accuracy improvement of the use of GDN layers with regard no using GDN layers is higher in the modified images because they present local contrast and luminance changes, the ones the GDN is supposed to deal with.

After the good performance in the classification problem we faced the main problem, the autonomous driving segmentation problem. Here, we trained three U-Net based models with four, one and any GDN layers with the Cityscapes dataset. Then, we tested the models not only in the original-good weather images but also in three fog level conditions, which introduced an important texture problem, such as the ones we

wanted to solve. The models with 4-GDN layers trained only with the no fog images showed a substantial increase in the intersection over union metric in all the weather conditions. In fact, the increase in the performance of the model gets progressively bigger when weather condition become worst, which shows that the higher level of fog the more important the GDN layers became. Also, as expected, for progressively heavier fog, the intersection over union metric gets reduced in all the models. However the conventional architecture (No-GDN layers) is more sensitive to the decrease in visibility (bigger reductions in performance) than the architecture with 4-GDN layers. Only with this we had already achieved the goal of the project showing that with GDN layers segmentation models improve their recognition and that GDN layer became more important for bad weather conditions. However, we got the predictions were we saw that GDN layers are so important that without them, in case of heavy fog, the models are not able to distinguish vehicles that are only a few metres away.

Finally, we also performed an analysis of the two first GDN layers of the 4-GDN model. We obtained that, as expected they boost low values and moderate high value and, more interesting, they helped to distinguish parts of the images that in the case of heavy fog would be indistinguishable without the GDN layers and made the models almost invariant to the fog.

To sum up, we achieved the objective of the Master Thesis and we showed that the use of GDN layers can help the models to be independent of changed in non-informative features, such as texture or contrast, both in classification and more interestingly in an autonomous driving segmentation problem.

# Chapter 5

## Extensions

During the time spent on this project, we have thought about many extensions that can be made. We have already started to do some of them in the context of the PhD, but we present all of them here in case anyone wants to extend the work done.

On one hand, one logical next step is to train the different models with more complicated images. In this point it would be interesting to check what happens when the shape of the images is not reduced. It is expected to improve the metrics because information is being lost when the shape is reduced. Moreover, the ideal would be to have not only one dataset, but to have millions of images from very different cities and countries and with a lot of variability. It means that, in the best scenario, we should have million of train images that should have been taken in all weather conditions (including snowing, cloudy, foggy and windy days), at all hours of the day (specially in the morning, afternoon and of course at night) and with many types of illumination (for example during the day, at night with and without streetlights, in tunnels or with the sun facing the camera). One option would be to include the coarse annotated images from Cityscapes, which will increase in 20000 the number of images available. Other interesting option would be train the models with also the fog images from the Foggy Cityscapes. Finally, it would be also important to search for new databases and train and test the models with new images.

Also, another step that should be done is to modify the models so that they can process video instead of just images. This process is necessary to implement this GDN models in autonomous vehicles. An improvement is expected when the video input will replace the images because in that case the model could use continuous monitoring. In that case, if for example a car is detected in one frame, it should be also in the next frame. So that, to improve the results, the models could be changed to receive as input not only the frame of the video but also the predictions that the model have perform in the previous frames. In this case, it would be trying to replicate a recurrent learning model through the use of video as input.

On the other hand, more models could be tested. For example it would be interesting to try different models architectures and with all the possible GDN layer combinations. It means to implement different models with different number of GDN layers and at different places. Moreover, another extension that can be done is to implement the GDN inverse layer in the expansive path of the U-Net in order to made the models completely symmetric. However, although it is an interesting test, we do not expected it to have a high impact in the final results because in this problem the models do not try to restore the image, they just want to predict the pixel class.

After that, the last step is to implement the model in a small autonomous robot, with for example a Google Coral, to test the models in real time, checking if it works as expected. Here, different experiments can be done, such as change the light or even put the robot in a dark room. Also, it would be interesting to point a flashlight to the robot to see if it dazzles or if it is able to adapt to this situation. If these tests are successfully completed, the possibility of speaking with a company to implement these models in their products could be considered.

Also, neural networks, and specially convolutional neural networks, try to replicate the human brain and so we expected them to have human behaviours. For example, human visual system shows adaptation to changes in the frequency of the stimuli [65, 66] and in order to replicate it, the models should exhibit this adaptation too. One logical extension of this project is to check if the trained models present this and other types of human-like adaptations and behaviours.

Finally, it is important to highlight that this Master Thesis will be continued with a PhD called “Aprendizaje Profundo Bio-Inspirado (Bio-Deep)”. During the next four years the use of this type of normalizations, its theoretical fundamentals and its extensions and applications will be investigated deeply. In fact, a summary with the results of only the segmentation models have been already submitted to the 2022 IEEE International Conference in Image Processing [67]. Some of the extensions mentioned above such us training with the fog images or testing human behaviours have already started to be done and when finish they will be also available in the same GitHub and, hopefully, in a paper. Therefore, this Master Thesis can be seen as a first approach to the field through a direct application.

# Chapter 6

## Bibliography

- [1] S. International, “Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles,” *SAE Mobilus*, 2018.
- [2] T. Kanade, C. Thorpe, and W. Whittaker, “Autonomous land vehicle project at cmu,” in *Proceedings of the 1986 ACM Fourteenth Annual Conference on Computer Science, CSC '86*, (New York, NY, USA), p. 71–80, Association for Computing Machinery, 1986.
- [3] “Automotive sensors – the sense organs of driver assistance systems.” <https://www.bmw.com/en/innovation/automotive-sensors.html>. Accessed: 2021-07-24.
- [4] “Platypus – r package for object detection and image segmentation.” <https://www.r-bloggers.com/2020/10/platypus-r-package-for-object-detection-and-image-segmentation/>. Accessed: 2022-04-12.
- [5] A. A. Novikov, D. Lenis, D. Major, J. Hladuvka, M. Wimmer, and K. Bühler, “Fully convolutional architectures for multiclass segmentation in chest radiographs,” *IEEE transactions on medical imaging*, vol. 37, no. 8, pp. 1865–1876, 2018.
- [6] N. A. Muhadi, A. F. Abdullah, S. K. Bejo, M. R. Mahadi, and A. Mijic, “Image segmentation methods for flood monitoring system,” *Water*, vol. 12, no. 6, 2020.
- [7] “Autocasion: Tesla, investigada en los ee.uu: sus coches con autopilot chocan contra vehículos de emergencia.” <https://www.autocasion.com/actualidad/noticias/tesla-investigada-autopilot-chocan-contra-vehiculos-emergencia>. Accessed: 2021-09-30.
- [8] Y. Bengio, “Learning deep architectures for AI,” *Fdns and Trends in ML*, vol. 2, no. 1, pp. 1–127, 2009.
- [9] D. Marini and A. Rizzi, “A computational approach to color adaptation effects,” *Image and Vision Computing*, vol. 18, no. 13, pp. 1005–1014, 2000.

- [10] S. Grossberg, “Nonlinear neural networks: Principles, mechanisms, and architectures,” *Neural networks*, vol. 1, no. 1, pp. 17–61, 1988.
- [11] K. I. Naka and W. A. H. Rushton, “S-potentials from luminosity units in the retina of fish (cyprinidae),” *The Journal of Physiology*, vol. 185, 1966.
- [12] D. J. Heeger, “Normalization of cell responses in cat striate cortex,” *Visual Neuroscience*, vol. 9, no. 2, p. 181–197, 1992.
- [13] M. Carandini and D. Heeger, “Normalization as a canonical neural computation,” *Nature Reviews Neuroscience*, vol. 13, pp. 51–62, 2012.
- [14] “Forbes: Tesla in taiwan crashes directly into overturned truck, ignores pedestrian, with autopilot on.” <https://www.forbes.com/sites/bradtempleton/2020/06/02/tesla-in-taiwan-crashes-directly-into-overturned-truck-ignores-pedestrian-with-autopilot-on/?sh=783fc01d58e5>. Accessed: 2021-09-30.
- [15] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Tech. Rep. 0, University of Toronto, Toronto, Ontario, 2009.
- [16] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The cityscapes dataset for semantic urban scene understanding,” in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [17] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.
- [18] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biological Cybernetics*, vol. 36, pp. 193–202, 2004.
- [19] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.
- [20] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [21] S. Zhang, L. Yao, A. Sun, and Y. Tay, “Deep learning based recommender system: A survey and new perspectives,” *ACM Comput. Surv.*, vol. 52, Feb. 2019.
- [22] R. Collobert and J. Weston, “A unified architecture for natural language processing: Deep neural networks with multitask learning,” in *International Conference on Machine Learning, ICML ’08*, (New York, NY, USA), p. 160–167, Association for Computing Machinery, 2008.

- [23] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, “Convolutional neural networks for time series classification,” *Journal of Systems Engineering and Electronics*, vol. 28, no. 1, pp. 162–169, 2017.
- [24] Z. Dai, H. Liu, Q. V. Le, and M. Tan, “Coatnet: Marrying convolution and attention for all data sizes,” *arXiv preprint arXiv:2106.04803*, 2021.
- [25] A. Tao, K. Sapra, and B. Catanzaro, “Hierarchical Multi-Scale Attention for Semantic Segmentation,” *CoRR*, 2020.
- [26] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *CoRR*, vol. abs/1603.07285, 2016.
- [27] “Towards data science: Understanding semantic segmentation with unet.” <https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>. Accessed: 2021-09-29.
- [28] “Towards data science: Transposed convolution demystified.” <https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba>. Accessed: 2021-09-30.
- [29] S. Haykin, *Neural networks and learning machines*. Upper Saddle River, NJ: Pearson Education, third ed., 2009.
- [30] T. Cohen and M. Welling, “Group equivariant convolutional networks,” in *International conference on machine learning*, pp. 2990–2999, PMLR, 2016.
- [31] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.
- [32] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?,” in *2009 IEEE 12th International Conference on Computer Vision*, pp. 2146–2153, 2009.
- [33] N. Rabinowitz, B. Willmore, J. Schnupp, and A. King, “Contrast gain control in auditory cortex,” *Neuron*, vol. 70, no. 6, pp. 1178–1191, 2011.
- [34] M. Martinez-Garcia, P. Cyriac, T. Batard, M. Bertalmío, and J. Malo, “Derivatives and inverse of cascaded linear+ nonlinear neural models,” *PloS one*, vol. 13, no. 10, p. e0201326, 2018.
- [35] A. Watson *et al.*, “Model of visual contrast gain control and pattern masking,” *JOSA*, vol. 14, no. 9, pp. 2379–2391, 1997.
- [36] M. Carandini and D. J. Heeger, “Summation and division by neurons in primate visual cortex,” *Science*, vol. 264, no. 5163, pp. 1333–1336, 1994.
- [37] M. P. Eckstein, A. J. Ahumada, and A. B. Watson, “Visual signal detection in structured backgrounds. ii. effects of contrast gain control, background variations, and white noise,” *JOSA A*, vol. 14, no. 9, pp. 2406–2419, 1997.

- [38] J. M. Hillis and D. H. Brainard, “Do common mechanisms of adaptation mediate color discrimination and appearance? uniform backgrounds,” *J. Opt. Soc. Am. A*, vol. 22, pp. 2090–2106, Oct 2005.
- [39] A. B. Abrams, J. M. Hillis, and D. H. Brainard, “The Relation Between Color Discrimination and Color Constancy: When Is Optimal Adaptation Task Dependent?,” *Neural Computation*, vol. 19, pp. 2610–2637, 10 2007.
- [40] A. Pons, J. Malo, J. Artigas, and P. Capilla, “Image quality metric based on multidimensional contrast perception models,” *Displays*, vol. 20, no. 2, pp. 93–110, 1999.
- [41] O. Schwartz and E. P. Simoncelli, “Natural signal statistics and sensory gain control,” *Nature neuroscience*, vol. 4, no. 8, pp. 819–825, 2001.
- [42] R. Coen-Cagli and O. Schwartz, “The impact on midlevel vision of statistically optimal divisive normalization in v1,” *Journal of vision*, vol. 13, no. 8, pp. 13–13, 2013.
- [43] J. Malo, I. Epifanio, R. Navarro, and E. Simoncelli, “Nonlinear image representation for efficient perceptual coding,” *IEEE Transactions on Image Processing*, vol. 15, no. 1, pp. 68–80, 2006.
- [44] J. Gutierrez, F. J. Ferri, and J. Malo, “Regularization operators for natural images based on nonlinear perception models,” *IEEE Transactions on Image Processing*, vol. 15, no. 1, pp. 189–200, 2005.
- [45] J. Ballé, V. Laparra, and E. P. Simoncelli, “Density modeling of images using a generalized normalization transformation,” in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
- [46] J. Ballé, V. Laparra, and E. P. Simoncelli, “End-to-end optimization of nonlinear transform codes for perceptual quality,” in *2016 Picture Coding Symposium (PCS)*, pp. 1–5, IEEE, 2016.
- [47] V. Laparra, J. Ballé, A. Berardino, and E. P. Simoncelli, “Perceptual image quality assessment using a normalized laplacian pyramid,” *Electronic Imaging*, vol. 2016, no. 16, pp. 1–6, 2016.
- [48] V. Laparra, A. Berardino, J. Ballé, and E. P. Simoncelli, “Perceptually optimized image rendering,” *J. Opt. Soc. Am. A*, vol. 34, pp. 1511–1525, Sep 2017.
- [49] J. Ballé, V. Laparra, and E. P. Simoncelli, “End-to-end optimized image compression,” in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.
- [50] M. F. Burg, S. A. Cadena, G. H. Denfield, E. Y. Walker, A. S. Tolia, M. Bethge, and A. S. Ecker, “Learning divisive normalization in primary visual cortex,” *PLOS Computational Biology*, vol. 17, no. 6, p. e1009028, 2021.



- [51] L. G. S. Giraldo and O. Schwartz, “Integrating flexible normalization into middle-level representations of deep convolutional neural networks,” *Neural Computation*, vol. 31, pp. 2138–2176, 2019.
- [52] X. Pan, L. G. S. Giraldo, E. Kartal, and O. Schwartz, “Brain-inspired weighted normalization for cnn image classification,” *bioRxiv*, 2021.
- [53] M. Martinez-Garcia *et al.*, “In praise of artifice reloaded: Caution with natural image databases in modeling vision,” *Frontiers in Neuroscience*, vol. 13, 2019.
- [54] A. Hepburn, V. Laparra, J. Malo, R. McConville, and R. Santos-Rodriguez, “Perceptnet: A human visual system inspired neural network for estimating perceptual distance,” in *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 121–125, IEEE, 2020.
- [55] M. Michelle *et al.*, “Divisive feature normalization improves image recognition performance in alexnet,” in *2022 ICLR*, 2022.
- [56] M. Ren *et al.*, “Normalizing the normalizers: Comparing and extending network normalization schemes,” 2017.
- [57] A. Watson and J. Malo, “Video quality measures based on the standard spatial observer,” in *Proceedings. International Conference on Image Processing*, vol. 3, pp. III–III, 2002.
- [58] V. Laparra, J. M. noz Marí, and J. Malo, “Divisive normalization image quality metric revisited,” *J. Opt. Soc. Am. A*, vol. 27, pp. 852–864, Apr 2010.
- [59] J. Malo, I. Epifanio, R. F. Navarro, and E. P. Simoncelli, “Nonlinear image representation for efficient perceptual coding,” *IEEE Transactions on Image Processing*, vol. 15, pp. 68–80, 2006.
- [60] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
- [61] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” *arXiv preprint arXiv:1708.07747*, 2017.
- [62] M. D. Fairchild, *Color appearance models*. John Wiley & Sons, 2013.
- [63] C. Sakaridis *et al.*, “Semantic foggy scene understanding with synthetic data,” *Int. J. Comput. Vis.*, vol. 126, no. 9, pp. 973–992, 2018.
- [64] R. Padilla, S. L. Netto, and E. A. da Silva, “A survey on performance metrics for object-detection algorithms,” in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242, IEEE, 2020.
- [65] F. W. Campbell and J. G. Robson, “Application of fourier analysis to the visibility of gratings,” *The Journal of Physiology*, vol. 197, 1968.

- [66] C. Blakemore and F. W. Campbell, “On the existence of neurones in the human visual system selectively sensitive to the orientation and size of retinal images,” *The Journal of Physiology*, vol. 203, 1969.
- [67] P. Hernández-Cámara, V. Laparra, and J. Malo, “Neural networks with divisive normalization for image segmentation with application in cityscapes dataset,” *ArXiv*, vol. abs/2203.13558, 2022.

# List of Figures

|      |  |    |
|------|--|----|
| 1.1  | Computer vision branches . . . . .                                   | 10 |
| 1.2  | The problem: variability due to non-informative factors . . . . .    | 11 |
| 2.1  | Convolution . . . . .  | 15 |
| 2.2  | Max pooling . . . . .  | 15 |
| 2.3  | Transposed convolution . . . . .                                     | 16 |
| 2.4  | The solution: divisive normalization manifold equalization . . . . . | 19 |
| 2.5  | Identity GDN test MNIST . . . . .                                    | 23 |
| 2.6  | Blurred GDN test MNIST . . . . .                                     | 24 |
| 2.7  | Blurred GDN test MNIST 2 . . . . .                                   | 25 |
| 2.8  | Blurred GDN test Fashion MNIST . . . . .                             | 26 |
| 2.9  | Classification models . . . . .                                      | 26 |
| 2.10 | U-Net architecture . . . . .   | 27 |
| 2.11 | Segmentation models . . . . .  | 29 |
| 3.1  | Classification dataset: Transformations . . . . .                    | 32 |
| 3.2  | Classification training curves . . . . .                             | 34 |
| 3.3  | Classification training difference curves . . . . .                  | 35 |
| 3.4  | Segmentation dataset: Image locations . . . . .                      | 37 |

---

|      |   |    |
|------|---|----|
| 3.5  | Segmentation dataset: Image and ground truth . . . . .          | 38 |
| 3.6  | Segmentation dataset: Image preprocess . . . . .                | 39 |
| 3.7  | Segmentation dataset: Cityscapes and Foggy Cityscapes . . . . . | 40 |
| 3.8  | Intersection over union . . . . .                               | 41 |
| 3.9  | Segmentation training difference curves . . . . .               | 44 |
| 3.10 | Segmentation predictions . . . . .                              | 45 |
| 3.11 | Segmentation GDN analysis . . . . .                             | 47 |

# List of Tables

|     |   |    |
|-----|---|----|
| 3.1 | Classification results . . . . .                              | 36 |
| 3.2 | Segmentation dataset: Cityscapes classes and colors . . . . . | 38 |
| 3.3 | Segmentation results . . . . .                                | 43 |