University of Padova

Department of Mathematics

Master Thesis in Data Science

Exploring Deep Learning Domain Adaptation performance: from Covariate Shift to Wasserstein and beyond

Supervisor
Professor Lamberto Ballan
University of Padova

Co-supervisor
Professor Étienne Baudrier
University of Strasbourg

Master Candidate
Marco Furlan

Academic Year

2023-2024

To my family, who always lovingly supported me, regardless of the struggles I faced or the distance I was.

Abstract

This thesis investigates the efficacy of Batch Normalization Adaptation, an unsupervised Domain Adaptation technique, in the context of image segmentation of mitochondria utilizing a UNet model. The primary objective is to empirically examine the performance of Batch Normalization Adaptation across different source and target datasets, to potentially predict its effectiveness in an unsupervised manner. Here, we illustrate the key findings derived from the pursuit of this objective.

Contents

Listing of figures

Listing of tables

Listing of acronyms

BN	Batch Normalization
IoU	Intersection over union

1 Introduction

From April to October 2023, I completed my internship at the ICube Laboratory at the University of Strasbourg. This thesis represents the culmination of my work during that period.

The objective of my Internship was as follows: "**given a U-Net model trained for the task of image segmentation (specifically to segment mitochondria in microscopic cell images), understand when the BatchNorm adaptation works and when it doesn't on different target datasets**". I will explain everything clearly below.

The **U-Net** is a convolutional neural network developed for biomedical image segmentation. It was created at the Computer Science Department of the University of Freiburg [2].

The datasets provided to me contain **microscopic cell images**, with the masks containing the segmentation of mitochondria. Refer to Figures 1.1 and 1.2 for two examples.

The idea is taking a U-Net trained on one source dataset, for example the dataset FS1 from Figure 1.1, and adapting it so that it is able to segment the images from a new target dataset, for example the dataset FS2 from Figure 1.2. This task in machine learning is known under the name of ***Domain Adaptation***. One simple and intuitive Domain Adaptation technique is the **BatchNorm Adaptation**, which will be explained in detail in Section 2.4.

My work is a continuation of the work of my team: specifically, I have leveraged the experiments presented in their article referenced as [1] as a foundational basis for my work. In the mentioned article, it was discovered that there is a relationship between the following two quantities:

- Wasserstein distance between the distributions of some toy datasets;
- the performance of the adapted U-Net.

Consequently, our hope was to find a relationship between:

- Wasserstein distance between the latent spaces of the U-Net and the adapted U-Net, both interpreted as probability distributions;
- the performance of the adapted U-Net.

Essentially we are looking for a predictor of the performance of the BatchNorm-Adapted U-Net.

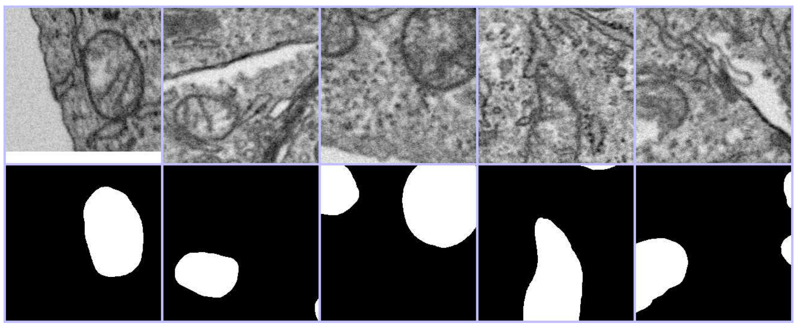


Figure 1.1: Dataset FS1 and its ground truth mitochondria segmentation. The dataset FS1 contains images obtained via chemical fixation.

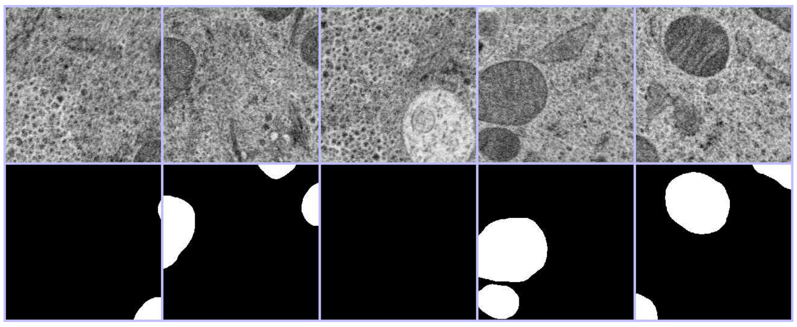


Figure 1.2: Dataset FS2 and its ground truth mitochondria segmentation. The dataset FS2 contains images obtained via cryo-fixation.

2 Theory

2.1 Distribution Shift and Domain Adaptation

Let us consider a labeled training dataset from a source distribution $p(\mathbf{x}, \mathbf{y})$ which we use to fit a predictive model $p(\mathbf{y}|\mathbf{x})$, and a test dataset from a target distribution $q(\mathbf{x}, \mathbf{y})$.

For context, in our case, $\mathbf{x}$ represents the images [*], $\mathbf{y}$ represents the masks [†]. As an example, if we set the dataset FS1 (Figure 1.1) as source and FS2 (Figure 1.2) as target, then $p(\mathbf{x}, \mathbf{y})$ is the distribution from which the images and masks of the dataset FS1 are generated, and $q(\mathbf{x}, \mathbf{y})$ is the distribution from which the images and masks of the dataset FS2 are generated, and the predictive model $p(\mathbf{y}|\mathbf{x})$ learns to predict the masks $\mathbf{y}$ from the images $\mathbf{x}$ of the dataset FS1.

A **distribution shift** or **dataset shift** occurs if $p \neq q$. In this case, the predictive model $p(\mathbf{y}|\mathbf{x})$ cannot be used to predict $q(\mathbf{y}|\mathbf{x})$, unless properly adapted.

There are four main types of distribution shift [3], which are summarised in Table 2.1:

- the **Covariate shift** or **Domain shift** occurs when the distribution of the images changes, while that of the masks stays the same. Example: the source dataset contains images of coffee pots, while the target dataset images of coffee pots with white noise;
- the **Concept shift** or **annotation shift** occurs when the distribution of the images is the same, but the masks (annotations) change. This can occur in case of different conven-

[*] so if RGB images it is $\mathbf{x} \in \{0, 1, ..., 255\}^{W \times H \times 3}$, also commonly written in papers such as [1] as the continuous generalization $\mathbf{x} \in \mathbb{R}^{W \times H \times 3}$

[†] so $\mathbf{y} \in \{0, 1\}^{W \times H}$

Discriminative: $X \rightarrow Y$

	$p(\mathbf{x}, \mathbf{y})$	$q(\mathbf{x}, \mathbf{y})$	$p(\mathbf{x}) \overset{?}{=} q(\mathbf{x})$	$p(\mathbf{y}\|\mathbf{x}) \overset{?}{=} q(\mathbf{y}\|\mathbf{x})$
Covariate shift	$p(\mathbf{x})p(\mathbf{y}\|\mathbf{x})$	$q(\mathbf{x})p(\mathbf{y}\|\mathbf{x})$	$\neq$	$=$
Concept shift	$p(\mathbf{x})p(\mathbf{y}\|\mathbf{x})$	$p(\mathbf{x})q(\mathbf{y}\|\mathbf{x})$	$=$	$\neq$

Generative: $X \leftarrow Y$

	$p(\mathbf{x}, \mathbf{y})$	$q(\mathbf{x}, \mathbf{y})$	$p(\mathbf{y}) \overset{?}{=} q(\mathbf{y})$	$p(\mathbf{x}\|\mathbf{y}) \overset{?}{=} q(\mathbf{x}\|\mathbf{y})$
Label shift	$p(\mathbf{y})p(\mathbf{x}\|\mathbf{y})$	$q(\mathbf{y})p(\mathbf{x}\|\mathbf{y})$	$\neq$	$=$
Manifestation shift	$p(\mathbf{y})p(\mathbf{x}\|\mathbf{y})$	$p(\mathbf{y})q(\mathbf{x}\|\mathbf{y})$	$=$	$\neq$

Table 2.1: The 4 types of distribution shift.

tions for annotating the images. Example: given a street segmentation task, two different organizations may or may not consider the sidewalk as a part of the class "street" (in many contexts it is in fact considered part of the street or roadway).

- the **Label shift** or **prior shift** occurs when the distribution of the label changes. For example, in medical imaging if we are trying to predict the presence of a disease, data gathered from a rural hospital may present a much diverse number of cases in which the disease is present compared to data from a urban hospital.
- the **Manifestation shift** occurs when the same label manifests in different images. For example, in medical imaging the presence of a tumor can manifest in different areas, so we may have a source images with brain tumor images and target images with lung tumor.

To deal with all these different types of domain shift, we introduce **Domain Adaptation**: formally, domain adaptation refers to the process of adapting a predictive model trained on data from one domain (**source**) to make accurate predictions on data from a different but related domain (**target**).

2.2 Batch Normalization

Batch Normalization (BatchNorm) is a technique commonly used in deep neural networks to stabilize and accelerate the training process. It's applied as a layer within a neural network, usually after the output of a convolutional or fully connected layer and before an activation function.

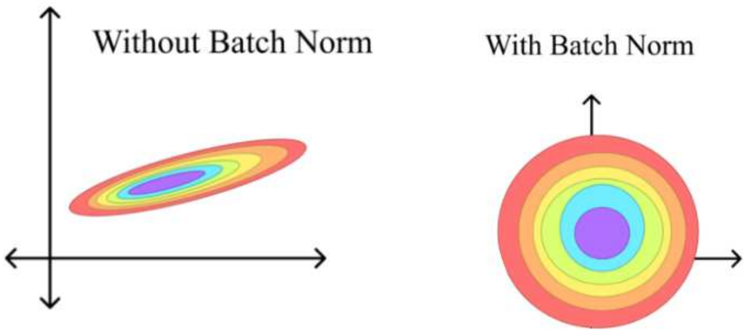


Figure 2.1: Simple visualization of the effect of Batch Normalization on a distribution.

The main purpose of Batch Normalization is to normalize the input of a layer by adjusting the mean and variance of the batch of data that passes through it (see Figure 2.1). This helps to mitigate the issue of **internal covariate shift** (see [4]), where the distribution of input data changes throughout the training process, causing slower convergence and requiring careful tuning of learning rates.

Computationally, this operation is realized as follows (from the official PyTorch implementation [5]):

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

Where x and y are 4-dimensional input and output, with the usual Torch tensor dimensions (N_{batch}, C, H, W) which are respectively number of images per-batch, number of channels (3 for RGB images), width and height; $E[x]$ and $\text{Var}[x]$ are running estimates of the mean and variance calculated per-dimension over mini-batches; ϵ is a value added for numerical stability (10^{-5} by default); γ and β are C-dimensional vectors of learnable parameters, initialized to 1 and 0 respectively.

The running estimates are calculated according to the formula:

$$\hat{x}_{new} = (1 - \text{momentum}) \times \hat{x} + \text{momentum} \times x_t$$

where $\hat{x}$ is the estimated statistic and x_t is the new observed value, and $\text{momentum} = 0.1$ by default.

For more details refer to the official PyTorch documentation [5].

2.3 U-Net

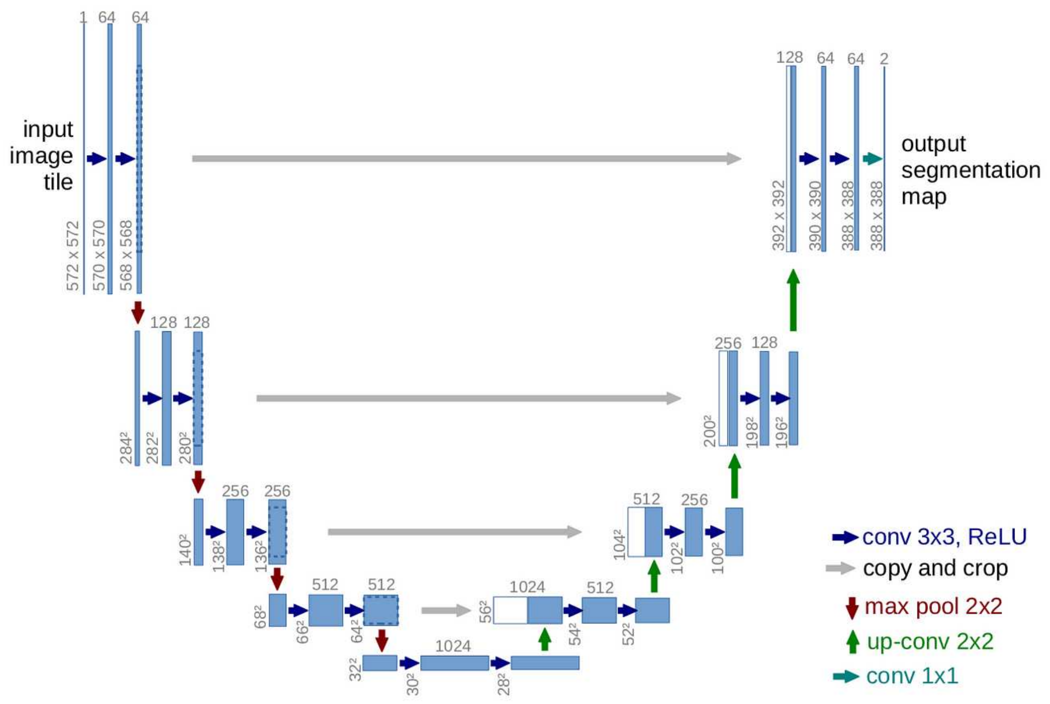


Figure 2.2: UNet structure from [2]. In our U-Net each blue arrow, which corresponds to *conv 3x3*, *ReLU*, has an additional BatchNorm layer: *conv 3x3*, *BatchNorm*, *ReLU*.

As already said in the introduction, the **U-Net** is a convolutional neural network developed for biomedical image segmentation. It was created at the Computer Science Department of the University of Freiburg [2]. It was named "U-Net" due to its U-shaped network architecture (Figure 2.2). It was originally developed for biomedical image segmentation, but its effectiveness has led to its use in various other domains as well.

The U-shaped architecture of the U-Net consists of a contracting path (encoder) and an expanding path (decoder). The contracting path captures context and reduces the spatial dimensions of the input image, while the expanding path recovers the spatial information and generates the segmented output. It is particularly effective for tasks where precise object boundaries need to be detected in images, such as medical image segmentation, where it's used to locate and segment structures like cells, organs, or tumors.

The architecture is characterized by skip connections, where the feature maps from the contracting path are combined with those in the expanding path. This helps in preserving fine details during the upsampling process.

Overall, U-Net has proven to be a powerful and widely used architecture for various image segmentation tasks in the field of machine learning and computer vision [6].

2.4 BatchNorm Adaptation

BatchNorm Adaptation is realized by taking a model already trained on the source images, and passing all target images through it, freezing all parameters but updating the running estimates $E[x]$ and Var$[x]$ in all BatchNorm layers. So the BatchNorm layers get adapted to the target images [7].

This is an **unsupervised** Domain Adaptation technique, meaning it does not require the target masks to be applied, just the target images. The implementation takes around 10 lines of code and the calculations a few seconds.

It was verified to better the performance significantly in certain cases, for example if the source dataset is FS1 (Figure 1.1) and the target dataset is FS2 (Figure 1.2) as shown in the article [1]; a sample image is shown in Figure 2.3.

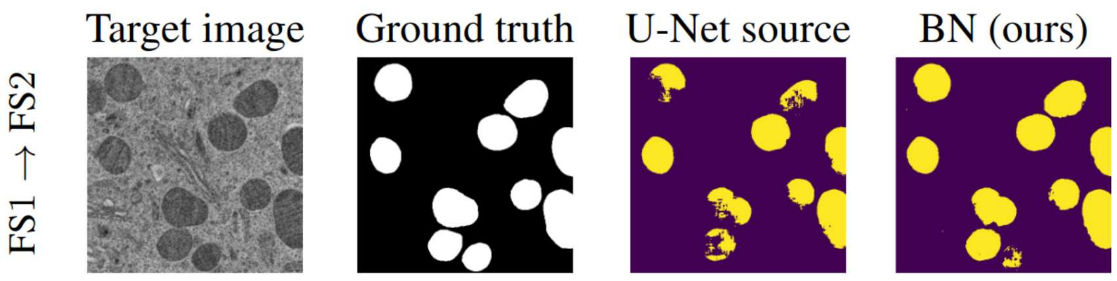


Figure 2.3: Image taken from Article [1]. As we can see the BatchNorm Adapted prediction (right-most image) is much better than the non-adapted U-Net (second image from the right). In this specific case the IoU goes from 0.556 (U-Net source) to 0.736 (U-Net BN-adapted).

2.5 Wasserstein Distance

Let (M, d) be a metric space that is a Radon space. For $p \in [1, \infty)$, the **Wasserstein p-distance** [8] between two probability measures μ and ν on M with finite p-moments is

$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \mathbb{E}_{(x,y)\sim\gamma} d(x, y)^p \right)^{1/p}$$

where $\Gamma(\mu, \nu)$ is the set of all couplings of μ and ν. A coupling γ is a joint probability measure on $M \times M$ whose marginals are μ and ν on the first and second factors, respectively. That is,

$$\int_M \gamma(x, y) dy = \mu(x)$$

$$\int_M \gamma(x, y) dx = \nu(y)$$

The Wasserstein distance is strictly related to the solution of the problem of optimal mass transportation; this gives a clean intuitive explanation of its definition [8].

The Wasserstein distance between two multivariate normal distributions,

$$d := W_2(N(m_1, \Sigma_1), N(m_2, \Sigma_2))$$

can be shown to have the following formula [‡] [10]:

$$d^2 = \|m_1 - m_2\|_2^2 + Tr(\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2} \Sigma_2 \Sigma_1^{1/2})^{1/2})$$

One of the most known applications of the Wasserstein distance is as a metric to assess the quality of images created by generative models such as GANs. In this context, it also is known under the name of **Fréchet Inception Distance** (see [11]). Specifically, the process in its most general form goes as the following pseudocode:

- **INPUT** a function $f : \Omega_X \to \mathbb{R}^n$
- **INPUT** two datasets $S, S' \subset \Omega_X$
- Compute $f(S), f(S') \subset \mathbb{R}^n$
- Fit two gaussian distributions $N(\mu, \Sigma), N(\mu', \Sigma')$, respectively for $f(S), f(S')$
- **RETURN** $W_2^2(N(\mu, \Sigma), N(\mu', \Sigma'))$

In the context of GANs, Ω_X is the space of images, and f is an Inception v3 model without its final classification layer (2048-dimensional activation vector).

In our case, Ω_X will be the space of images, and f will be a section of the U-Net (generally in the contracting part).

[‡]the square root of a matrix is not uniquely defined, so for clarity sake we specify that $\Sigma^{1/2}$ refers to the **principal square root matrix** [9]. Since we are dealing with covariance matrices, which are real, symmetric, positive semidefinite matrices, their principal square root matrix is also real, symmetric and positive semidefinite [9].

2.5.1 Source-Normalized Wasserstein and Target-Normalized Wasserstein

The **source-normalized Wasserstein distance** [12] normalizes the means and covariance matrices with respect to the source statistics. It is defined as follows:

$$
\begin{aligned}
{}^{sn}W_2^2\left(N(\mu_s,\Sigma_s),N(\mu_t,\Sigma_t)\right) = W_2^2\left(N(\Sigma_s^{-1/2}\mu_s,\mathbb{I}),N(\Sigma_s^{-1/2}\mu_t,\Sigma_s^{-1}\Sigma_t)\right) = \\
= (\mu_t-\mu_s)^T\Sigma_s^{-1}(\mu_t-\mu_s) + Tr\left(\mathbb{I}+\Sigma_t\Sigma_s^{-1}-2\Sigma_t^{-1/2}\Sigma_s^{-1/2}\right)
\end{aligned}
$$

Where the subscripts s and t refer to source and target distributions, and $\Sigma^{-1/2}=(\Sigma^{1/2})^{-1}$ is the inverse of the square root matrix of the covariance matrix Σ. Similarly, the **target-normalized Wasserstein distance** normalizes the means and covariance matrices with respect to the target statistics:

$$
{}^{tn}W_2^2\left(N(\mu_s,\Sigma_s),N(\mu_t,\Sigma_t)\right) = {}^{sn}W_2^2\left(N(\mu_t,\Sigma_t),N(\mu_s,\Sigma_s)\right)
$$

The explicit formula is therefore the same as above, swapping the subscripts s and t.

3 The Objective

3.1 The objective of my research Internship

As established in the Introdution, all the results in this Thesis are finalized for the following goal: **"given a U-Net model trained for the task of image segmentation (specifically to segment mitochondria in microscopic cell images), understand when the BatchNorm adaptation works and when it doesn't on different target datasets"**.

More in detail, we are hoping to find a relationship between the following two quantities:

- The **BatchNorm adapted U-Net performance**,
- the **Wasserstein distance** computed on the latent spaces of the original U-Net and the BatchNorm adapted U-Net.

This would allow to know in an unsupervised way if it's worth to apply the BatchNorm adaptation to our U-Net.

3.2 Motivation

This idea is shown on an experiment from the paper [1]. A neural network is used to conduct a basic binary classification task (see Figure 3.1). In this demonstration, a relationship is found

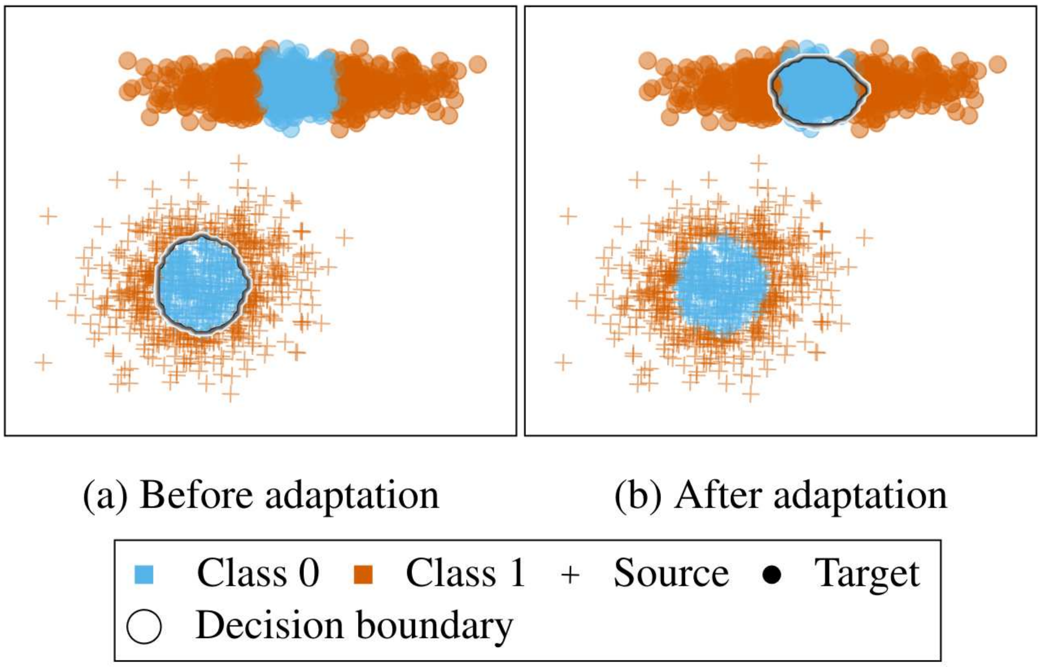


Figure 3.1: The binary classification task.

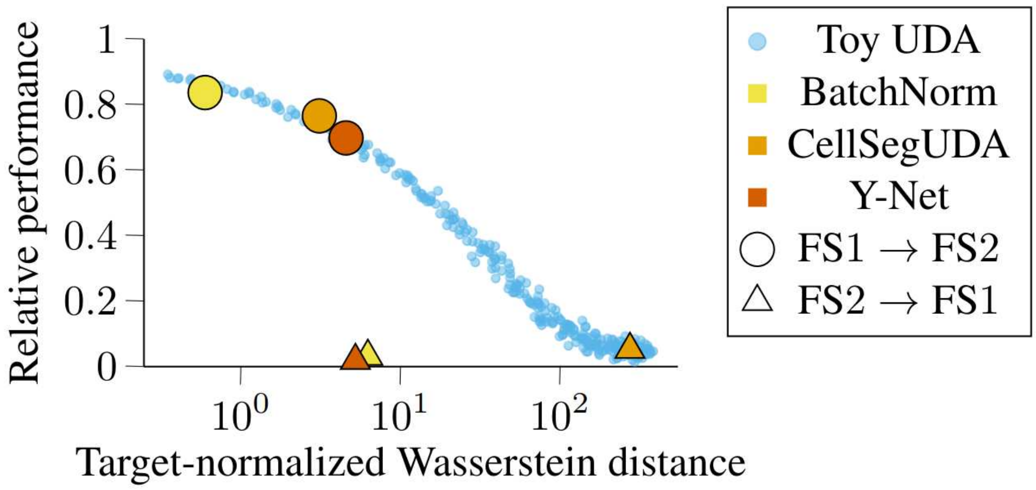


Figure 3.2: The relationship between Relative Performance P_{target}/P_{source} and Target-normalized Wasserstein distance (see [13]). The light blue points ToyUDA are different iteration of the experiment shown in Figure 3.1, where data was deformed according to different means and covariance matrices.

between the performance of a simple BN-adapted network and the Target-Normalized Wasserstein Distance [13] (see Figure 3.2).

Given the results of this experiment, the hope would be to find such a relationship in the more complex case of an image segmentation task. Specifically, we expect that after BN adaptation **the latent space(s) of the adapted U-Net will be similar to the latent space(s) of the original U-Net**, but only if the adapted U-Net performs well on the target. The idea is that if the features are similar after BN adaptation, then also the feature spaces are similar.

Therefore, we need to compute a distance between the latent spaces of original U-Net and adapted U-Net. To do so, we proceed as follows: we take the latent spaces of the original U-Net and the target adapted U-Net, consider them as two sets of samples from two distinct probability distibutions, and compute the (target-normalized) Wasserstein distance between them.

The reason for this is that BN adaptation is a relatively simple modification to the network. So our hope is that if BN adaptation is effective the two latent spaces will be similar, hence the Wasserstein distance will be lower. On the other hand, if it is not, the Wasserstein distance should be higher. Essentially, we are looking for behavior similar to that from the experiment (Figure 3.2).

Some experiments were already done by my team with the old toy datasets. Results were diverse (see Figure 3.3), but that far not promising yet. This was our starting point.

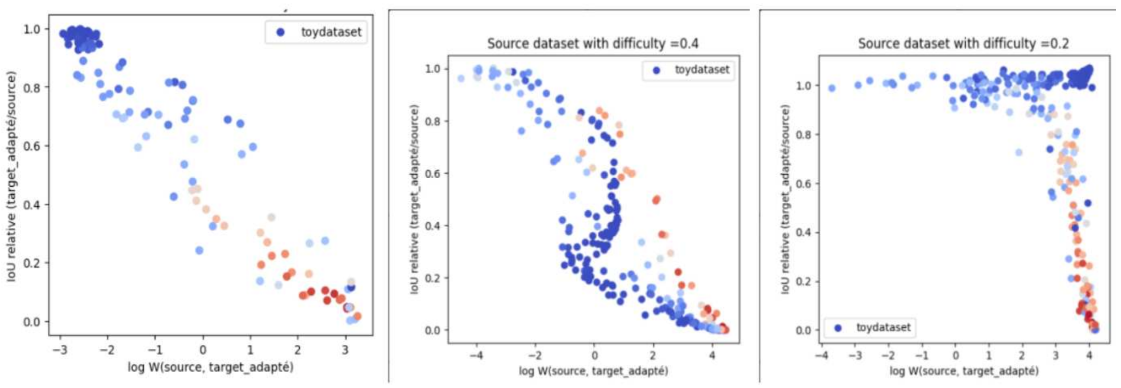


Figure 3.3: Some of the results are from experiments conducted by my team. The details of these experiments and the coloring of the dots are not relevant for our purposes; however, we can see that the only graph that provides some faint hope for a consistent pattern is the first one.

4
Generating toy datasets

To conduct new experiments big amounts of data were needed. Hence the necessity for an algorithm to generate datasets.

4.1 The old dataset generator: Bézier curves

The code generates points randomly in 2d and then connects them with a Bézier curve, creating the boundary of the toy mitochondria. Then the region inside the curve is defined to be white, and the outside black; this way the mask is created. Then, by replacing 0 and 1 pixels with two random normal white noises $N(\mu_0, \sigma_0)$ and $N(\mu_1, \sigma_1)$ we get the toy dataset; some samples are shown in Figure 4.1.

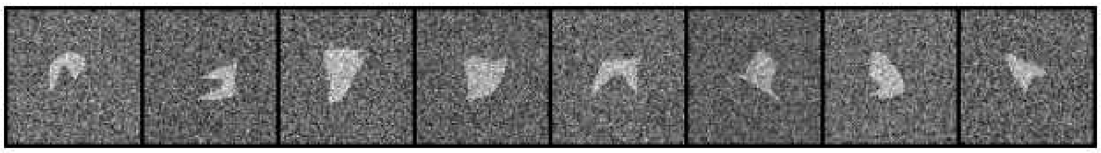


Figure 4.1: Example of dataset created with the old toy dataset generator, using Bézier curves.

This dataset generation has some clear drawbacks: it is intricate to code thus slow, and generates only one cell at a time which makes it hard to generalize.

4.2 The new dataset generator: Perlin noise

I decided to redo from scratch the dataset generation. My objective was to create 0/1 masks resembling the ones from the datasets FS1 and FS2 form Figures 1.1 and 1.2. The process goes as follows: first a 2d Perlin noise is generated (original code from [14]), then we set a threshold: all values whose modulus is above this threshold are 1, otherwise they are 0. Then as before we replace 0's and 1's with two random normal white noises $N(\mu_0, \sigma_0)$ and $N(\mu_1, \sigma_1)$.

A sample is shown in Figure 4.2, the process is summed up in Figure 4.3.

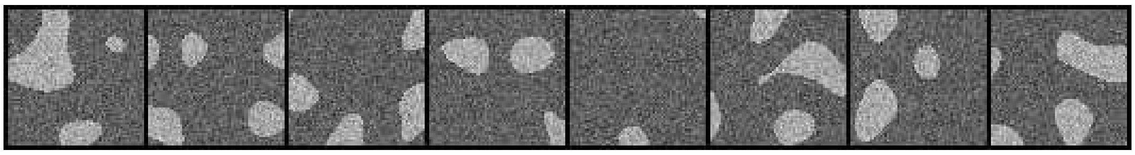


Figure 4.2: Examples of images created with the new toy dataset generator, using Perlin noise. These images are 64x64 pixels but they can be generated with any width and height.

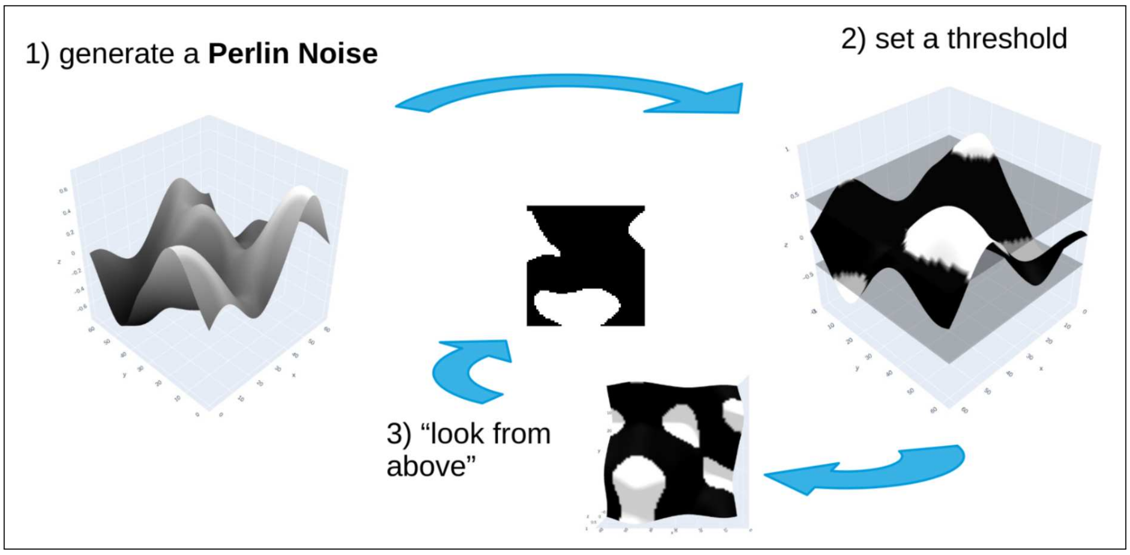


Figure 4.3: New dataset generation process.

Pros: This process is overall simpler and much faster than the previous algorithm, allowing to generate a dataset with 10.000 images and masks in around 40 seconds, with a random number of mitochondrias with different shapes, sizes and positions.

Cons: The shapes of course are similar to mitochondria to our eye but ultimately their distribution is different than true mitochondria masks. For example, shapes tend to be clearly detached from each other, while real mitochondria can get close to each other and almost touch. This is accounted for: in fact we are trying to get results out of the segmentation process, so all we need is a decent approximation of the original datasets, not a perfect procedural reconstruction.

4.2.1 More technical details

This subsection is dedicated to clarify some details about the dataset generation.

The translation correction

The Perlin noise has some constant zero-crossings: "[...] the [Perlin] noise function will pass through zero at every node, giving Perlin noise its characteristic look." (from [15]). This means that if we just generate a 64x64 Perlin noise we will end up with the zero-crossings in the same points, and the masks will always avoid these points (see Figures 4.4 and 4.5 on the left). This may not a problem since we are dealing with a fully convolutional network hence invariant by translation of the input, but it does alter the shapes created and it's generally good practice to not have such an unbalance in our dataset. To make up for this instead of generating a 64x64 Perlin Noise with 2x2 resolution we generate a 96x96 Perlin noise with 3x3 resolution and we randomly choose a 64x64 square in it (see Figure 4.4). This randomizes the position of the zero-crossings. With this modification the density of the masks across the dataset images is homogeneous (see Figure 4.5 on the right).

Why not use the original mitochondria masks instead?

The experimentations require massive amounts of data and when I started I had only two datasets to work with, namely FS1 and FS2 shown in Figures 1.1 and 1.2. The total number of masks of the two combined is 11.500 + 10.000, and a huge number of them are irrelevant (segmentations of very similar images, hence very similar masks). On the other hand, my algorithm can generate a full dataset with 10.000 images, all completely new and different from each other, in around 40 seconds, making it an obvious better choice, so me and my team decided to opt for this.

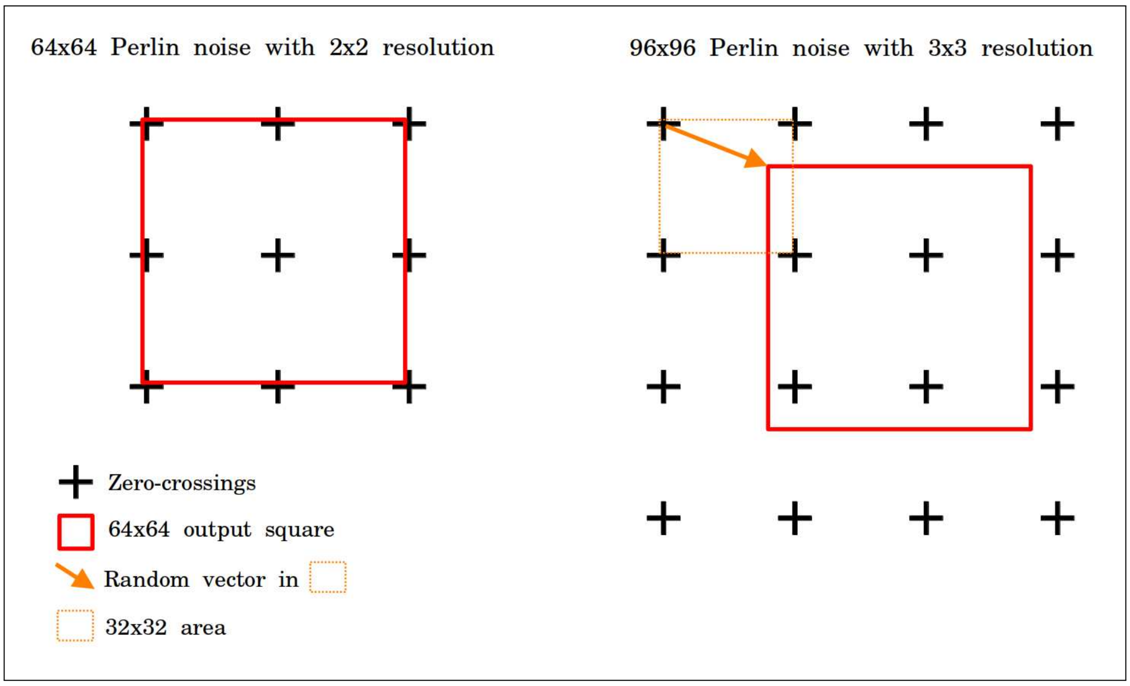

64x64 Perlin noise with 2x2 resolution
96x96 Perlin noise with 3x3 resolution
Zero-crossings
64x64 output square
Random vector in
32x32 area

Figure 4.4: Visual explanation of the translation correction. On the left, zero-crossings are always in the same position relative to the red square. On the right, the position of the red square is translated by a random amount between 0 and 31 on the x and y axis, so the zero-crossings are in random positions.

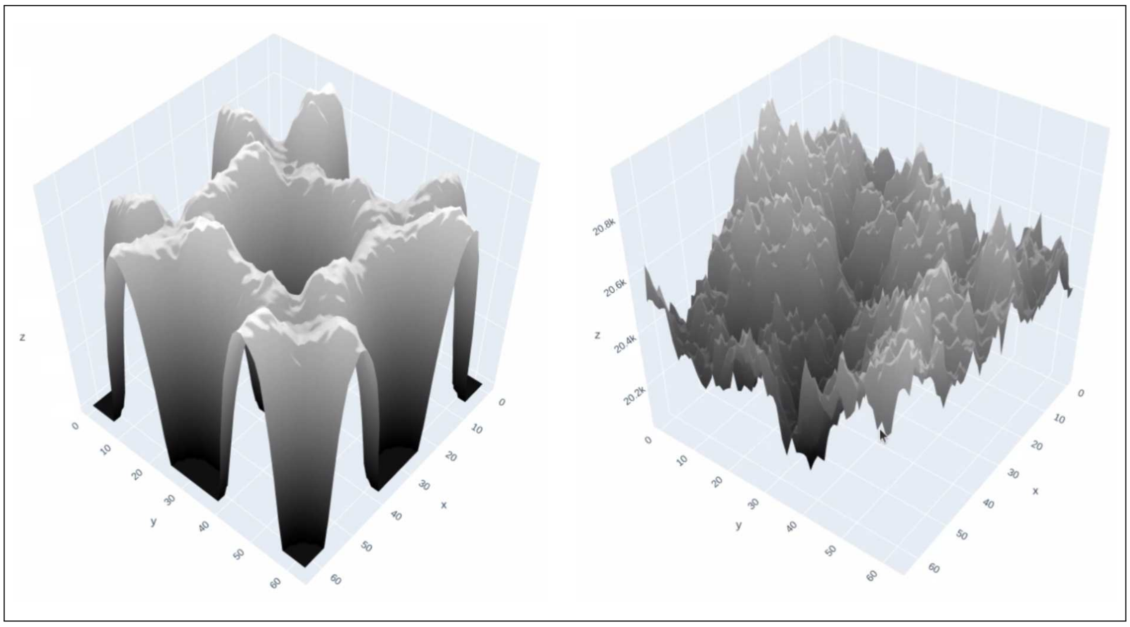


Figure 4.5: Density distribution of the white mask pixels before and after adding the translation. On the left, before correcting, we can see how the masks are completely absent in the black gorges, which correspond to the zero-crossings (z-axis ranges from 0 to >30.000). On the right, after correcting, the density is more homogeneous (z-axis ranges from 20.000 to 21.000).

5 Experiments

Several experiments were conducted. In this thesis we present a selection of them, those which we considered to be the most important.

5.1 Some starting results

Consider a dataset such as the one shown in Figure 4.2. A dataset of this kind has 4 parameters we can work with: $(\mu_0, \sigma_0, \mu_1, \sigma_1)$, which are the parameters of the background white noise $N(\mu_0, \sigma_0)$ and the parameters of the masks white noise $N(\mu_1, \sigma_1)$. We began with some simple experiments to establish baseline results.

5.1.1 Experiment #1: Increasing distance

As a first result, 10 sets of parameters were chosen. We kept $\sigma_0 = \sigma_1 = 50$ and changed μ_0, μ_1 in such a way that $\frac{1}{2}(\mu_0 + \mu_1) = 128$ and the distance $\mu_1 - \mu_0$ increased from dataset to dataset (see Figure 5.2). More specifically, the parameters were chosen this way:

$$\begin{aligned}(\mu_0, \mu_1) \in \{&(123, 133), (121, 135), (117, 139), (113, 143), (106, 150),\\ &(97, 159), (83, 173), (65, 191), (38, 218), (1, 255)\}\end{aligned}$$

Hence:

$$\mu_1 - \mu_0 \in \{10, 14, 22, 30, 44, 62, 90, 126, 180, 254\}$$

For each of these parameter choices we generated a source dataset and a target dataset, totalling 20 datasets. We then trained 10 models on the 10 source datasets, and used them to predict the 10 target datasets; then we computed the average IoU [*] between the ground truth masks and the predicted masks, and we plotted it (see Figure 5.3 on the left).

Afterwards, we took each of the 10 models and adapted to each of the 10 target datasets; this gave us 100 adapted models, which we used to predict the target datasets, get the average IoUs between ground truth masks and predicted masks, and plotted them (see Figure 5.3 on the right).

The results are shown in Figure 5.3.

Details: 5000 source images [†], 500 test images, batch size = 1, 5 epochs, early stopping with patience 5 [‡] and δ [§] $= 0$, percentage of validation set = 10%, learning rate = 10^{-5}, no image scaling, RMSprop optimizer (weight decay 10^{-8}, momentum 0.999, gradient clipping 1.0) with ReduceLROnPlateau scheduler. Loss function = CrossEntropyLoss + DiceLoss.

Observations:

- The first evident observation is the **irregular behavior** observed when the source dataset has $\mu_1 - \mu_0 = 30$ in the non-adapted model (Figure 5.3, on the left). Indeed, upon repeating this experiment or similar ones multiple times, such irregularities arose sporadically across different datasets. The reasons for this are difficult to infer, as they do not consistently arise, but we came up with some possible explanations: different convergence during training, variations in dataset generation, or *excessive clipping* when $\mu_1 - \mu_0 \geq 90$, which we will discuss later. **BN adaptation consistently corrects these types of irregularities**.

- Irregularities apart, the overall **results are consistent with our basic intuition**: from left to right the IoU increases (from hardest to easiest); training on a harder dataset (lower

[*]Intersection over union, common measure to quantify the goodness of a prediction in image segmentation. In our case it is computed on the white masks.

[†]The train-validation split is done automatically by the training code of the U-Net

[‡]Because of the way the code is in [16], 5 times per epochs the accuracy is computed on the validation set; this means that in 5 epochs there are 25 steps where the accuracy is checked. We programmed the early stopping on these steps.

[§]if the current accuracy is lower than the best accuracy plus δ, the early stopping counter is reset to zero

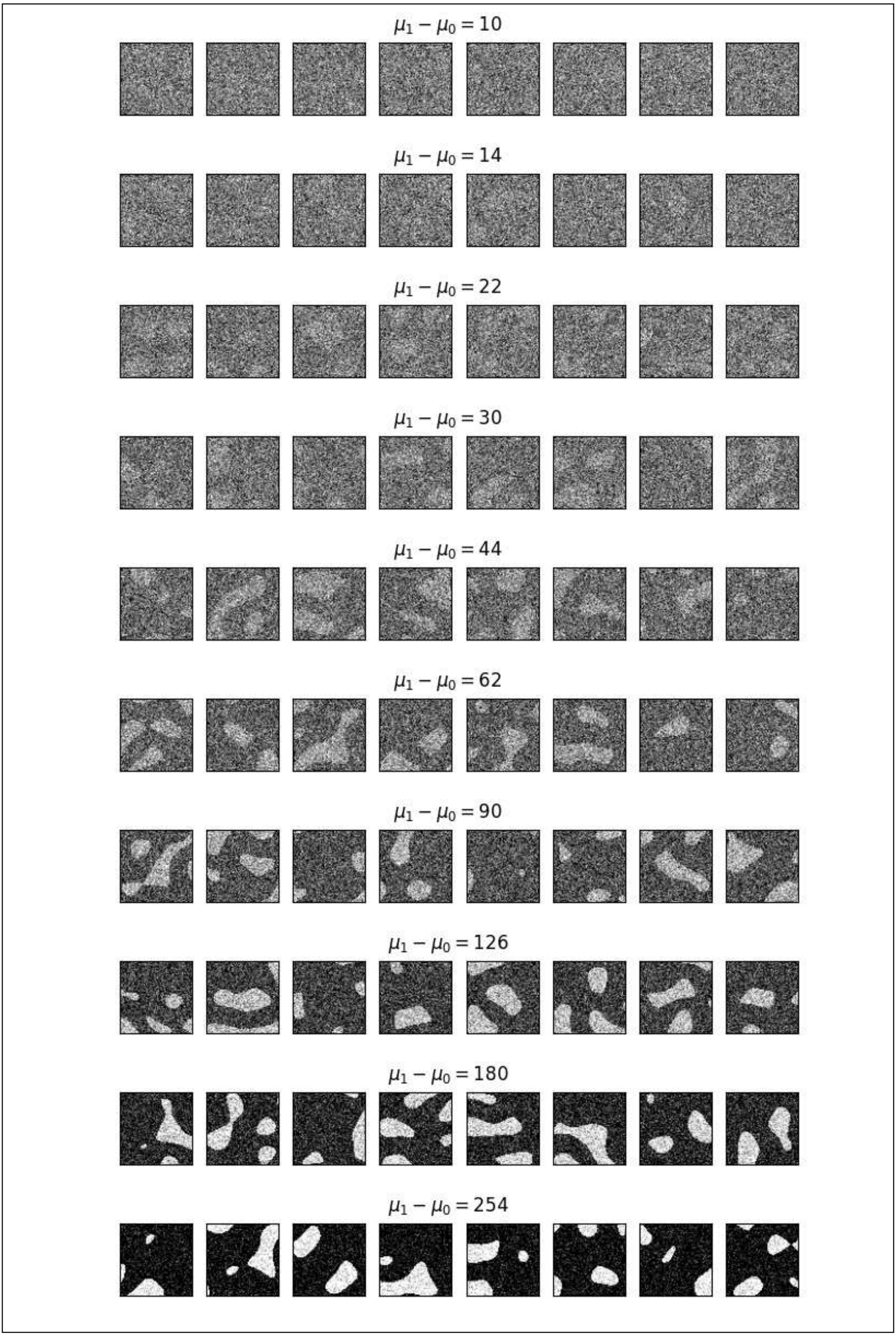


Figure 5.1: Samples of Experiment #1.

$\mu_1 - \mu_0$) and testing on an easier dataset (higher $\mu_1 - \mu_0$) generally results in better results than the opposite way; in the diagonal (where source $\mu_1 - \mu_0$ is equal to target $\mu_1 - \mu_0$) the IoU is consistently good, with no irregularities arising in any of the iterations of this

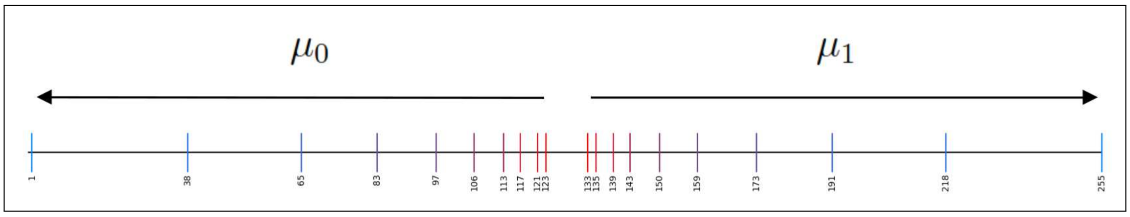


Figure 5.2: μ_0, μ_1 of Experiment #1. Same colors means same dataset, from red, $\mu_1 - \mu_0 = 10$ (hardest dataset) to blue, $\mu_1 - \mu_0 = 255$ (easiest dataset).

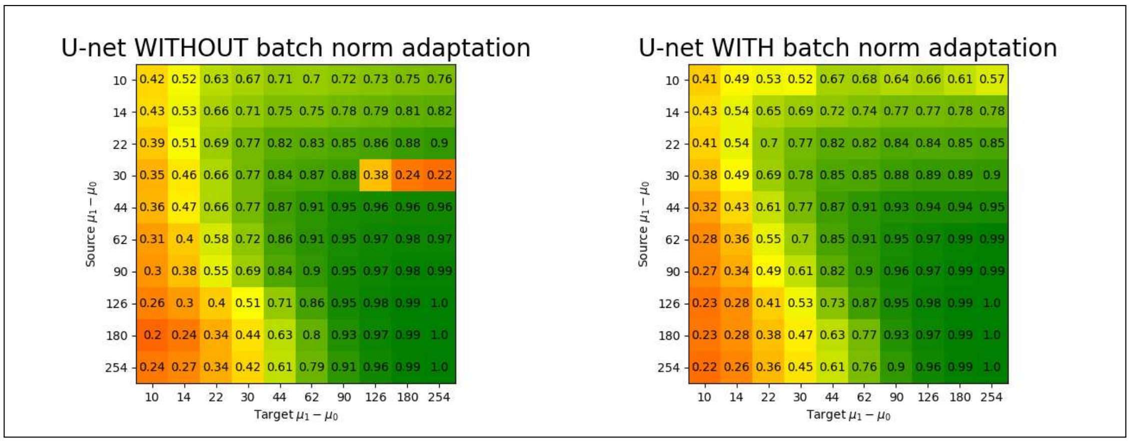


Figure 5.3: IoU graphs, without and with BatchNorm adaptation. The x-axis ticks show the $\mu_1 - \mu_0$ of the source dataset and the y-axis ticks the $\mu_1 - \mu_0$ of the target dataset.

experiment.

5.1.2 Experiment #2: brightness shift

9 sets of parameters were chosen. We kept $\sigma_0 = \sigma_1 = 50$ and changed μ_0, μ_1 in such a way that $\frac{1}{2}(\mu_0 + \mu_1) = 128$ and the distance $\mu_1 - \mu_0 = 40$ remained constant. More specifically, the parameters were chosen this way:

$$(\mu_0, \mu_1) \in \{(10, 50), (30, 70), (50, 90), (70, 110),$$
$$(90, 130), (110, 150), (130, 170), (150, 190), (170, 210)\}$$

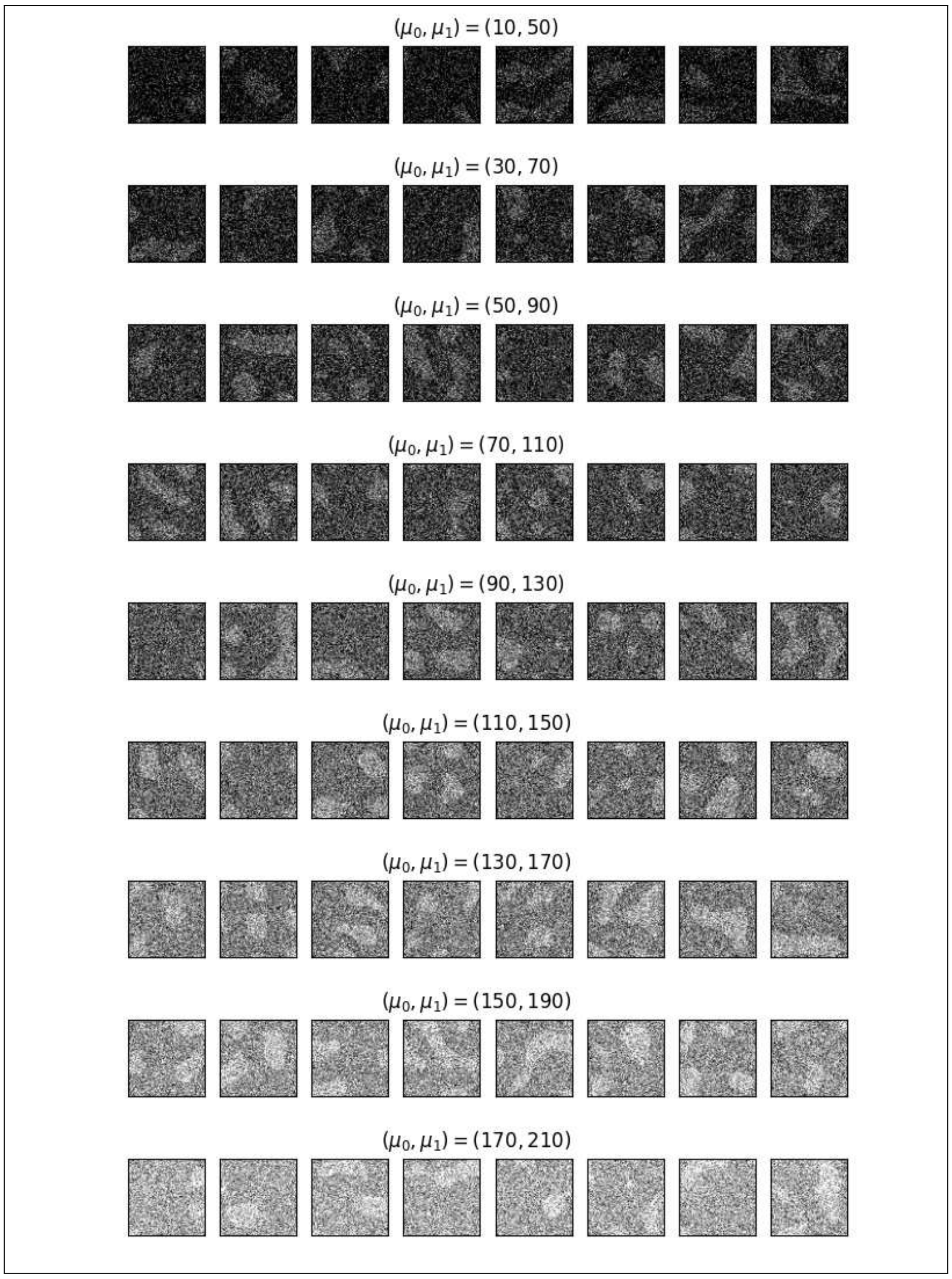

$(\mu_0, \mu_1) = (10, 50)$
$(\mu_0, \mu_1) = (30, 70)$
$(\mu_0, \mu_1) = (50, 90)$
$(\mu_0, \mu_1) = (70, 110)$
$(\mu_0, \mu_1) = (90, 130)$
$(\mu_0, \mu_1) = (110, 150)$
$(\mu_0, \mu_1) = (130, 170)$
$(\mu_0, \mu_1) = (150, 190)$
$(\mu_0, \mu_1) = (170, 210)$

Figure 5.4: Samples of Experiment #2.

The resulting datasets show what we describe intuitively as a "*brightness shift*" (See Figure 5.4). The results are shown in Figure 5.5.

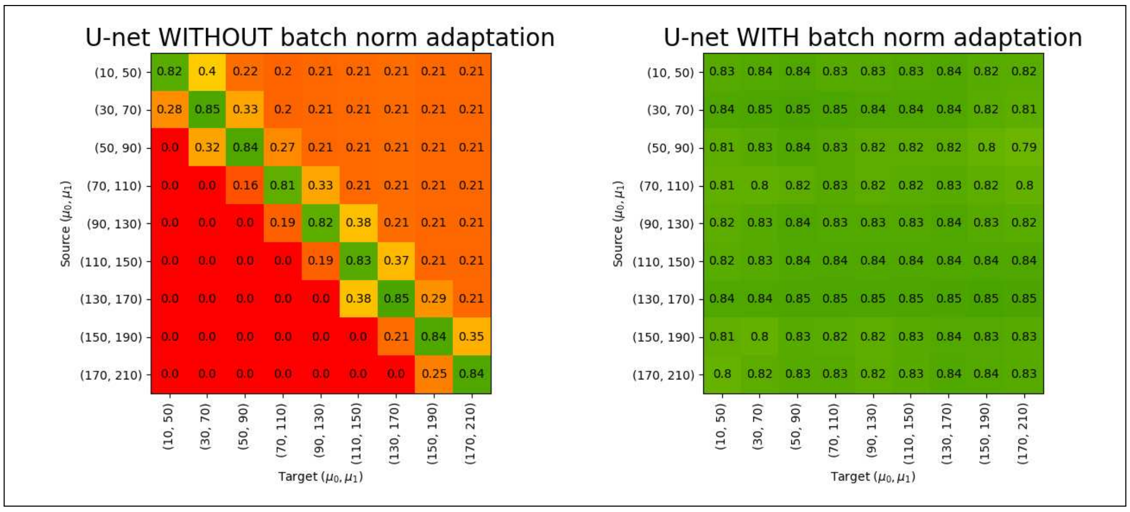


Figure 5.5: IoU graphs, without and with BatchNorm adaptation. The x-axis ticks show the $\mu_1 - \mu_0$ of the source dataset and the y-axis ticks the $\mu_1 - \mu_0$ of the target dataset.

Details: 5000 source images [¶], 500 test images, batch size = 1, 5 epochs, early stopping with patience 5 [‖] and δ [**] $= 0$, percentage of validation set = 10%, learning rate = 10^{-5}, no image scaling, RMSprop optimizer (weight decay 10^{-8}, momentum 0.999, gradient clipping 1.0) with ReduceLROnPlateau scheduler. Loss function = CrossEntropyLoss + DiceLoss.

Observations:

- Results of the non-adapted U-Net (Figure 5.5) are exactly what you would expect. To understand the reason for them, see Figure 5.6: as you can see, a net trained on a given source dataset will predict all black on darker target datasets, and all white on brighter target datasets. Hence 0.2 IoU on full white predictions (average number of white pixels of the masks [††]) and 0.0 IoU on full black predictions.

- The BatchNorm adaptation fully corrects for brightness shift. This is in fact a rather trivial result, since BN-adaptation shifts the running mean $\hat{\mu}$ from that of the source

[¶] The train-validation split is done automatically by the training code of the U-Net

[‖] Because of the way the code is in [16], 5 times per epochs the accuracy is computed on the validation set; this means that in 5 epochs there are 25 steps where the accuracy is checked. We programmed the early stopping on these steps.

[**] if the current accuracy is lower than the best accuracy plus δ, the early stopping counter is reset to zero

[††] If we assume images are independent samples from a distribution p it is then expected by the strong law of large number that the density of white pixels converges (as the number of samples tends to infinity).

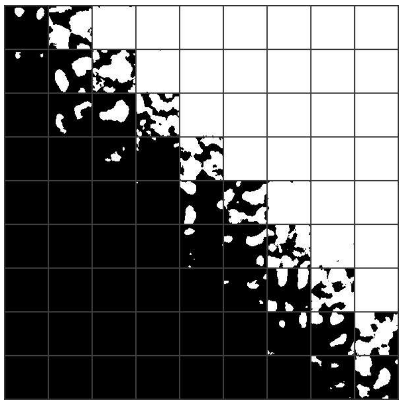


Figure 5.6: The predicted masks of the U-Net **without BN adaptation** in each of the source-target combinations. The plot grid follows the same pattern as the graphs in Figure 5.5. As intuitive, if source is darker and target is brighter the prediction is fully white, while it is fully black in the opposite case, hence the pattern that emerges in Figure 5.5.

dataset to that of the target dataset. Nonetheless, it is a great showcase of a scenario where this domain adaptation technique excels; this also generalizes to RGB images for each of the colors separately (as an example, we may have two datasets where the photos were taken with different devices which have different sensibilities to the three colors, resulting in three distinct "brightness shifts"-alike behaviours).

5.1.3 Experiment #3: mixed datasets

This experiment considers a wider set of datasets and combinations to account for more generality. Specifically we have:

- 4 datasets with increasing $\mu_1 - \mu_0$ distance, as in Experiment #1,
- 4 datasets with brightness shift, as in Experiment #2,
- 3 miscellaneous datasets: 1 random dataset, where all parameters $(\mu_0, \sigma_0, \mu_1, \sigma_1)$ are randomized, 1 gradient dataset, where 2 distinct color patterns are applied to mask and background, and 1 edges dataset, where we convolve the mask with an edge detection kernel to get the images.

Samples can be seen in Figure 5.7. The results are shown in Figure 5.8.

Details: 5000 source images , 500 test images, batch size = 1, 5 epochs, early stopping with patience 5 and $\delta = 0$, percentage of validation set = 10%, learning rate = 10^{-5}, no image scaling, RMSprop optimizer (weight decay 10^{-8}, momentum 0.999, gradient clipping 1.0) with ReduceLROnPlateau scheduler. Loss function = CrossEntropyLoss + DiceLoss.

Observations:

- Referring to **Figure 5.8**: we have drawn 3 subdivisions to separate the 3 groups of datasets, as in the dotted list above. The top-left and central 4x4 squares show the behaviour we've seen already from Experiments #1 and #2. The last square (bottom right, 3x3 square) shows the three new datasets and how they interact between each others as source and target datasets.

- A lot of different interactions are shown from these graphs and underlining each one separately would require a couple pages. The most interesting cases are where the BN-adaptation improves notably the performance in a source-target couple, but switching source and target leads to completely different results (see center-bottom rectangle and right-center rectangles, 3x4 and 4x3 respectively). Another very interesting example is how the U-Net trained on the *gradient* dataset is also successful in predicting the *edges* dataset even without any adaptation; on the other hand, the U-Net trained on the *edges* dataset performs terribly on the *gradient* dataset: our assumption is that in the first case the filters take the shape of a general edge-detection-like matrix, while in the second case the filters are more specialized.

- As in the example just mentioned above, this experiment shows plenty of cases where there is **source-target asymmetry**, which was one of the key observations from the paper of my team [1].

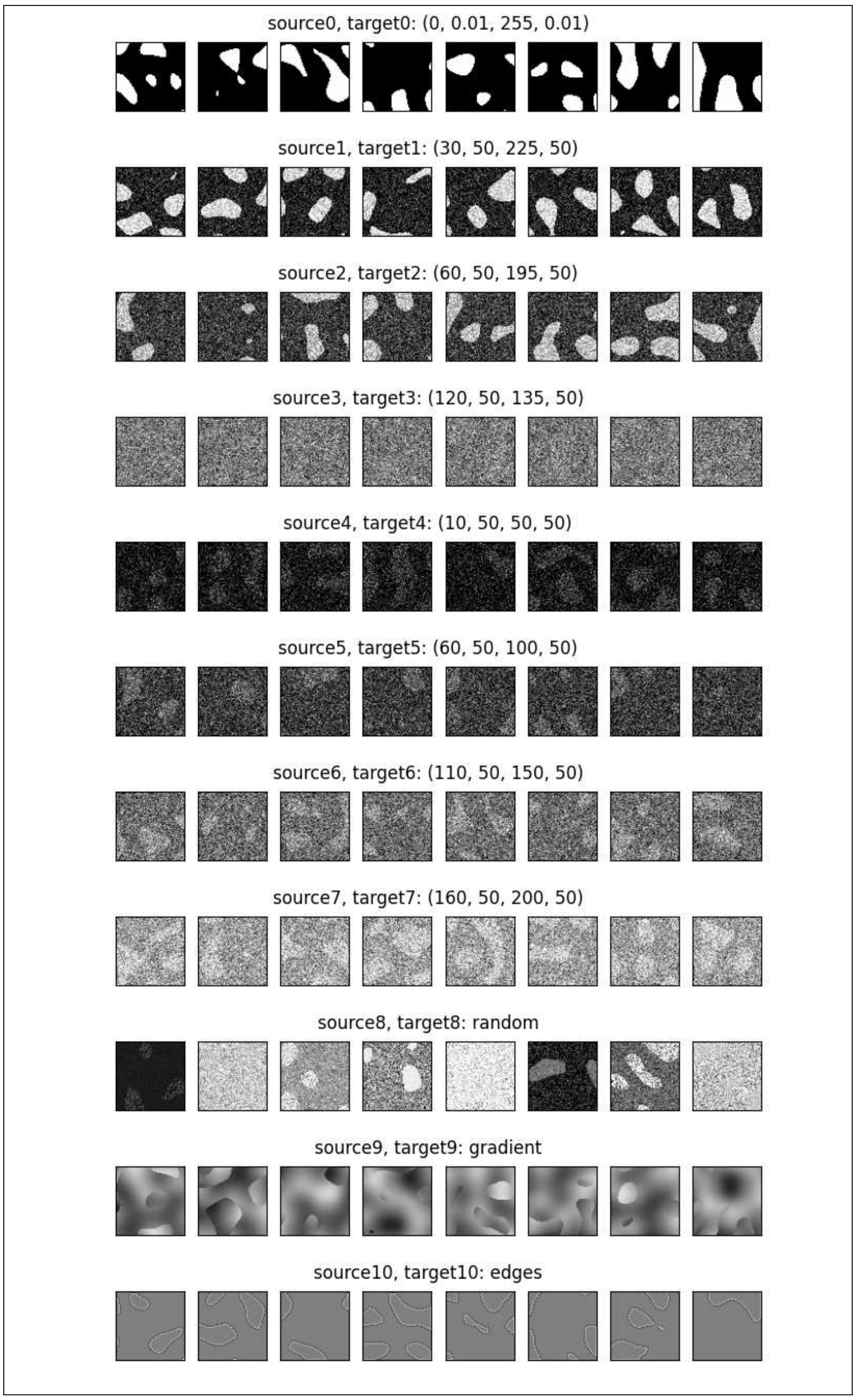
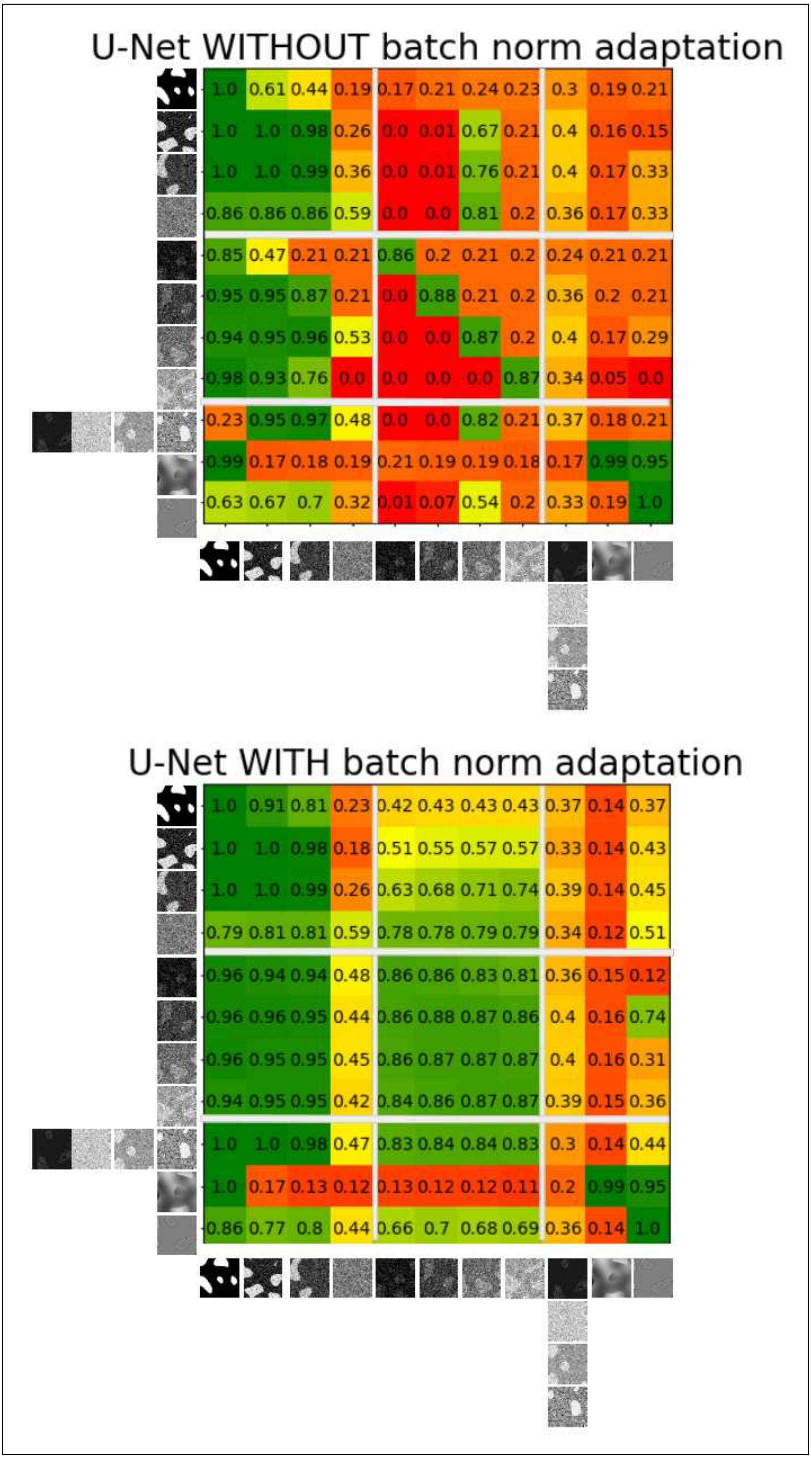

source0, target0: (0, 0.01, 255, 0.01)
source1, target1: (30, 50, 225, 50)
source2, target2: (60, 50, 195, 50)
source3, target3: (120, 50, 135, 50)
source4, target4: (10, 50, 50, 50)
source5, target5: (60, 50, 100, 50)
source6, target6: (110, 50, 150, 50)
source7, target7: (160, 50, 200, 50)
source8, target8: random
source9, target9: gradient
source10, target10: edges

Figure 5.7: Samples of Experiment #3.



U-Net WITHOUT batch norm adaptation

1.0	0.61	0.44	0.19	0.17	0.21	0.24	0.23	0.3	0.19	0.21
1.0	1.0	0.98	0.26	0.0	0.01	0.67	0.21	0.4	0.16	0.15
1.0	1.0	0.99	0.36	0.0	0.01	0.76	0.21	0.4	0.17	0.33
0.86	0.86	0.86	0.59	0.0	0.0	0.81	0.2	0.36	0.17	0.33
0.85	0.47	0.21	0.21	0.86	0.2	0.21	0.2	0.24	0.21	0.21
0.95	0.95	0.87	0.21	0.0	0.88	0.21	0.2	0.36	0.2	0.21
0.94	0.95	0.96	0.53	0.0	0.0	0.87	0.2	0.4	0.17	0.29
0.98	0.93	0.76	0.0	0.0	0.0	0.0	0.87	0.34	0.05	0.0
0.23	0.95	0.97	0.48	0.0	0.0	0.82	0.21	0.37	0.18	0.21
0.99	0.17	0.18	0.19	0.21	0.19	0.19	0.18	0.17	0.99	0.95
0.63	0.67	0.7	0.32	0.01	0.07	0.54	0.2	0.33	0.19	1.0

U-Net WITH batch norm adaptation

1.0	0.91	0.81	0.23	0.42	0.43	0.43	0.43	0.37	0.14	0.37
1.0	1.0	0.98	0.18	0.51	0.55	0.57	0.57	0.33	0.14	0.43
1.0	1.0	0.99	0.26	0.63	0.68	0.71	0.74	0.39	0.14	0.45
0.79	0.81	0.81	0.59	0.78	0.78	0.79	0.79	0.34	0.12	0.51
0.96	0.94	0.94	0.48	0.86	0.86	0.83	0.81	0.36	0.15	0.12
0.96	0.96	0.95	0.44	0.86	0.88	0.87	0.86	0.4	0.16	0.74
0.96	0.95	0.95	0.45	0.86	0.87	0.87	0.87	0.4	0.16	0.31
0.94	0.95	0.95	0.42	0.84	0.86	0.87	0.87	0.39	0.15	0.36
1.0	1.0	0.98	0.47	0.83	0.84	0.84	0.83	0.3	0.14	0.44
1.0	0.17	0.13	0.12	0.13	0.12	0.12	0.11	0.2	0.99	0.95
0.86	0.77	0.8	0.44	0.66	0.7	0.68	0.69	0.36	0.14	1.0

Figure 5.8: IoU of Experiment #3, before and after BN adaptation. On the y axis the source dataset, on the x axis the target dataset.

5.2 Latent Space Analysis

As we anticipated, we want to calculate the distance between the latent spaces of the original U-Net and the BatchNorm-adapted U-Net. Here we explain precisely how that is done, and all the issues encountered.

We assume our dataset contains RGB images of size 64x64. It is common in deep learning to consider images as samples from a probability distribution; specifically, such an input image would be a sample $x \in \mathbb{R}^{3\times 64\times 64}$ with a certain probability $\tilde{p}(x)$.

Firstly, we analysed the latent space for seeing what information it was bringing. The distribution is 1024-dimensional which is tricky to handle. As a first thing we checked the 1-dimensional projections: some very unusual behaviour was happening in some cases, for instance the one shown in Figure 5.9. After some modifications (considering before ReLU and removing weight decay) we got consistent regular behaviour (see Figure 5.10).

We wanted to check for potential Gaussian behaviour [17], but this was clearly not the case as shown in Figure 5.11.

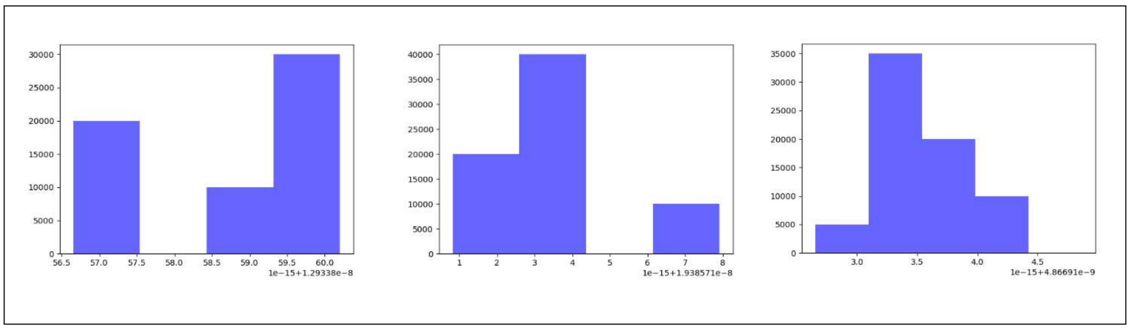


Figure 5.9: From Experiment #1, three 1d projections of the 1024-dimensional distribution, before the corrections. The source dataset is the one from Figure 5.1 with $\mu_1 - \mu_0 = 22$.

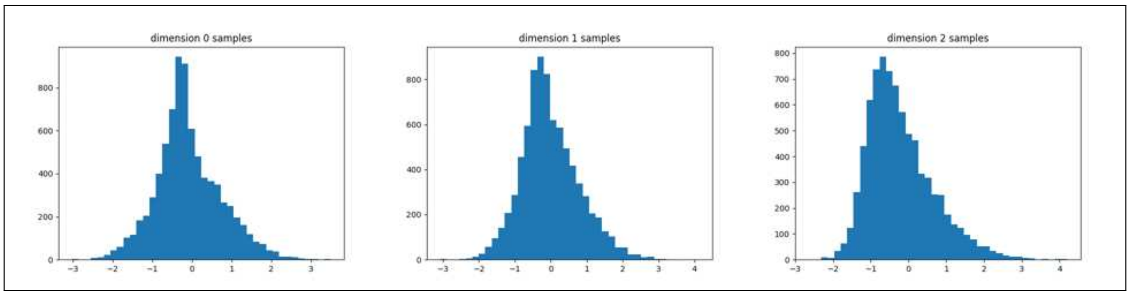


Figure 5.10: From Experiment #1, three 1d projections of the 1024-dimensional distribution, now with the corrections. Behaviour is regular and easier to manage.

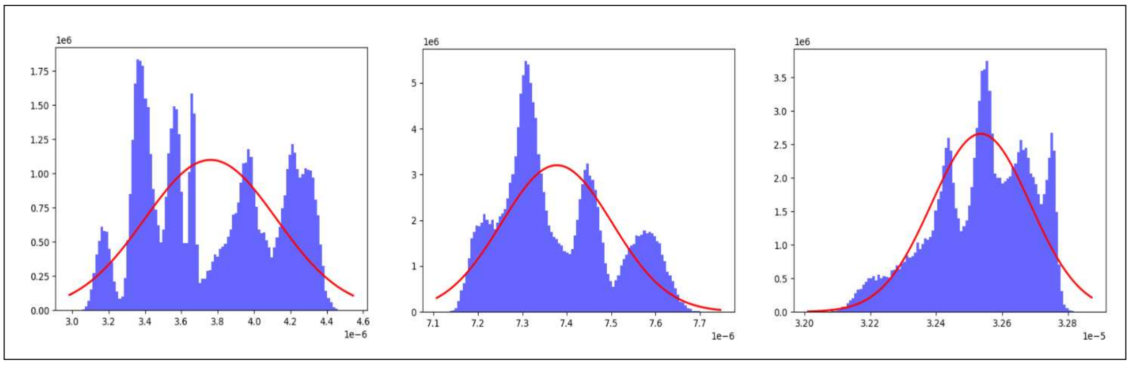


Figure 5.11: Three 1d projections of the 1024-dimensional distribution. The source dataset is the one from Figure 5.1 with $\mu_1 - \mu_0 = 10$. The red lines are fitted Gaussian: as we can see the distribution is not gaussian in this case.

5.2.1 Wasserstein accuracy

After testing the Wasserstein measure in a couple experiments, we realized that the computed values were relatively stable on the upper-most latent spaces, while they got unstable and unreliable on deeper layers, with a much wider span of values for the same computation and sporadically negative values (which is not something we want to see when computing any distance). Consequently, we set up a test to get an estimate of how accurate the Wasserstein formula is in function of the number of samples m and the dimensionality n. Our implementation of the Wasserstein was taken from the stable version of the Fréchet Distance proposed by PyTorch [18] [19].

The experiment goes as follows: we generate m samples from two identical n-dimensional Normal Gaussian distributions $N(0, I_n)$, $N(0, I_n)$, and we compute the squared Wasserstein distance between these two sets of samples, which ideally should be equal to 0.

Results are shown in Figure 5.12. A polynomial function $f_{err}(m, n)$ was fitted for all three curves and was used to infer the expected error size for $n \geq 100$. So the function $f_{err}(m, n)$

repesent the error size of the squared Wasserstein distance for m samples between two multivariate normal n-dimensional distributions.

We want to apply these results to the latent spaces of the U-Net, to get a rough idea of what the error size could be for each of our latent space. In fact, the deeper we go into the contractive path of our U-Net, the less samples and the higher the dimensionality (m decreases and n increases, both resulting in $f_{err}(m, n)$ increasing). Results are summed up in Table 5.1.

These results essenitally show that Wasserstein distance may be unreliable in the last two latent spaces **ls3** and **ls4**, while for the other three they are expected to be more reliable. Of course this is nothing but an example experiment, so the actual results may be different, but as we will see empirically they to line up with this idea.

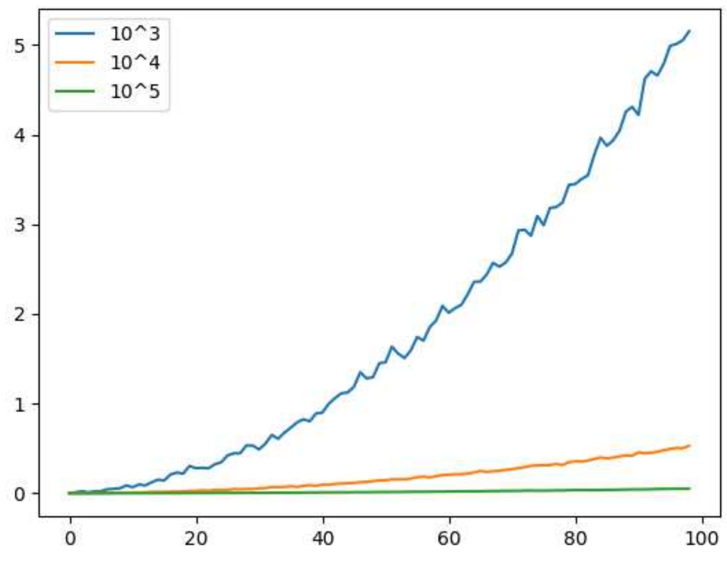


Figure 5.12: x-axis is $n \in [1, 99]$, y-axis is squared Wasserstein, legend is $m \in \{10^3, 10^4, 10^5\}$.

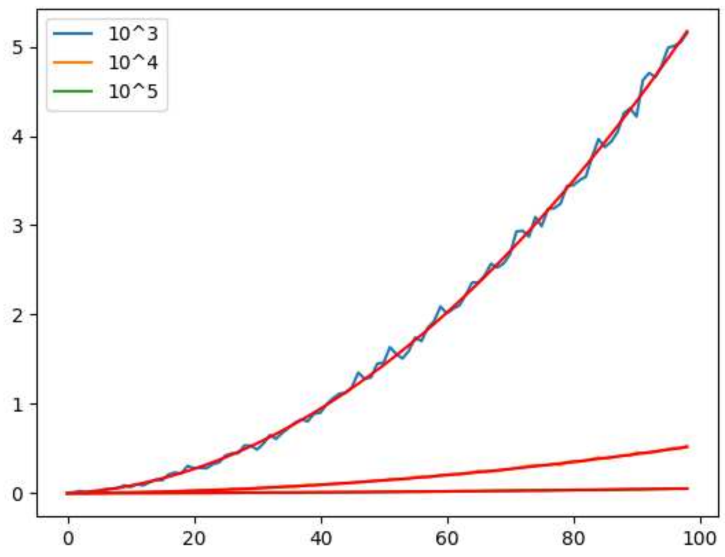


Figure 5.13: In red the functions $f_{err}(m,n)$ for $m = 10^3, 10^4, 10^5$. The exact function is $f_{err}(m,n) = \frac{1}{m}(0.502n^2 + 2.596n - 4.457)$.

	ls=0	**ls=1**	**ls=2**	**ls=3**	**ls=4**
$W \times H$	64×64	32×32	16×16	8×8	4×4
m	4096000	1024000	256000	64000	16000
n	64	128	256	512	1024
$f_{err}(m,n)$	$< \mathbf{10^3}$	$< \mathbf{10^3}$	**0.13**	**2.08**	**33**

Table 5.1: We assume 1000 input images, hence $m = 1000 \times W \times H$. $f_{err}(m,n)$ represents the error size of the squared Wasserstein calculation. **ls=0**, **ls=1** etc. are the latent spaces as shown in Figure 5.14.

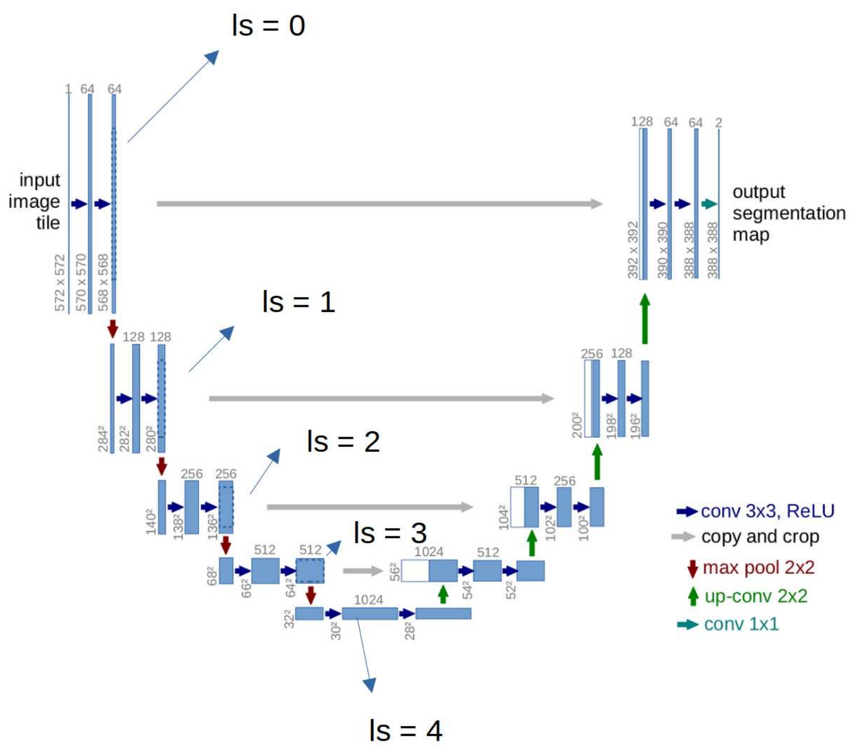


Figure 5.14: U-Net latent spaces.

5.3 The clipping issue

RGB images have values from 0 to 255 for each channel, but when the two white noises $N(\mu_0, \sigma_0)$ and $N(\mu_1, \sigma_1)$ are generated values range from $-\infty$ to ∞. So for values < 0 and > 255 we apply clipping to 0 and 255 respectively. This visually does not appear to be an issue, as shown in Figure 5.15, but it significantly affected the results. Consequently, we imposed for the new experiments the following constraints:

- $64 \leq \mu \leq 192$
- $0 < \sigma \leq 30$

This guarantees that clipping involves $< 2\%$ of the pixels [‡‡].

[‡‡]considering the expected number of clipped pixels.

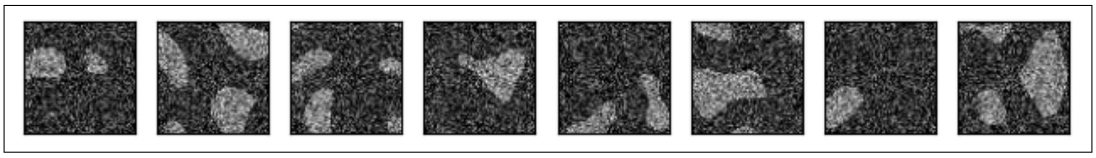


Figure 5.15: Some samples from a toy dataset with parameters $(\mu_0, \sigma_0, \mu_1, \sigma_1) = (50, 50, 130, 50)$. Despite looking regular and homogeneously noisy, these images have on average more than 10% of pixel values equal to 0 (full black). This is because if we define $X \sim N(\mu_0, \sigma_0)$ and call $p_X(x)$ its probability distribution function, then $\int_{-\infty}^{0} p_X(x)dx \approx 0.158 > 15\%$ and the background on average takes 80% of the images, resulting in at least $15\% \times 80\% = 12\%$ of the pixels of the images expected to be clipped to 0.

5.4 THE 1-800 EXPERIMENT

In this experiment, a single source model was trained on a dataset with parameters $(96, 18, 138, 18)$. It was then used to predict 800 target datasets, both before and after BN adaptation. The target datasets have the following parameters:

- $\mu_0 \in \{64, 80, 96, 112, 128, 144, 160\}$
- $\mu_1 \in \{\mu_1 + d : d \in \{24, 32, 42, 55, 73, 97, 128\} \land \mu_1 + d \leq 192\}$
- $\sigma_0 \in \{6, 12, 18, 24, 30\}$
- $\sigma_1 \in \{6, 12, 18, 24, 30\}$

This totals to 32 possible combinations of (μ_0, μ_1) and 25 possible combinations of (σ_0, σ_1), totalling 32x25 = 800 target datasets. All parameter combinations respect the constraints above.

Details: 5000 source images, 1000 test images, batch size = 1, 5 epochs, early stopping with patience 5 and $\delta = 0$, percentage of validation set = 10%, learning rate = 10^{-5}, no image scaling, RMSprop optimizer (weight decay 0, momentum 0.999, gradient clipping 1.0) with ReduceLROnPlateau scheduler. Loss function = CrossEntropyLoss + DiceLoss.

Observations:

- The results before BN adaptation (Figure 5.17) mirror our previous intuitions: values of 0.0 where prediction is full black, 0.2 where prediction is full white, and a gradient in between.
- We can see that BN adaptation works very well in all cases regardless of the movement of the parameters, which is a much more surprising result than the one shown in the brightness experiment from before. In this case in fact we are not shifting the entire image's mean, but we are shifting the background and masks' means and variances independently.

- This experiment is cleanly setup for a proper latent spaces analysis. We will refer to it for getting the main result of this Thesis.

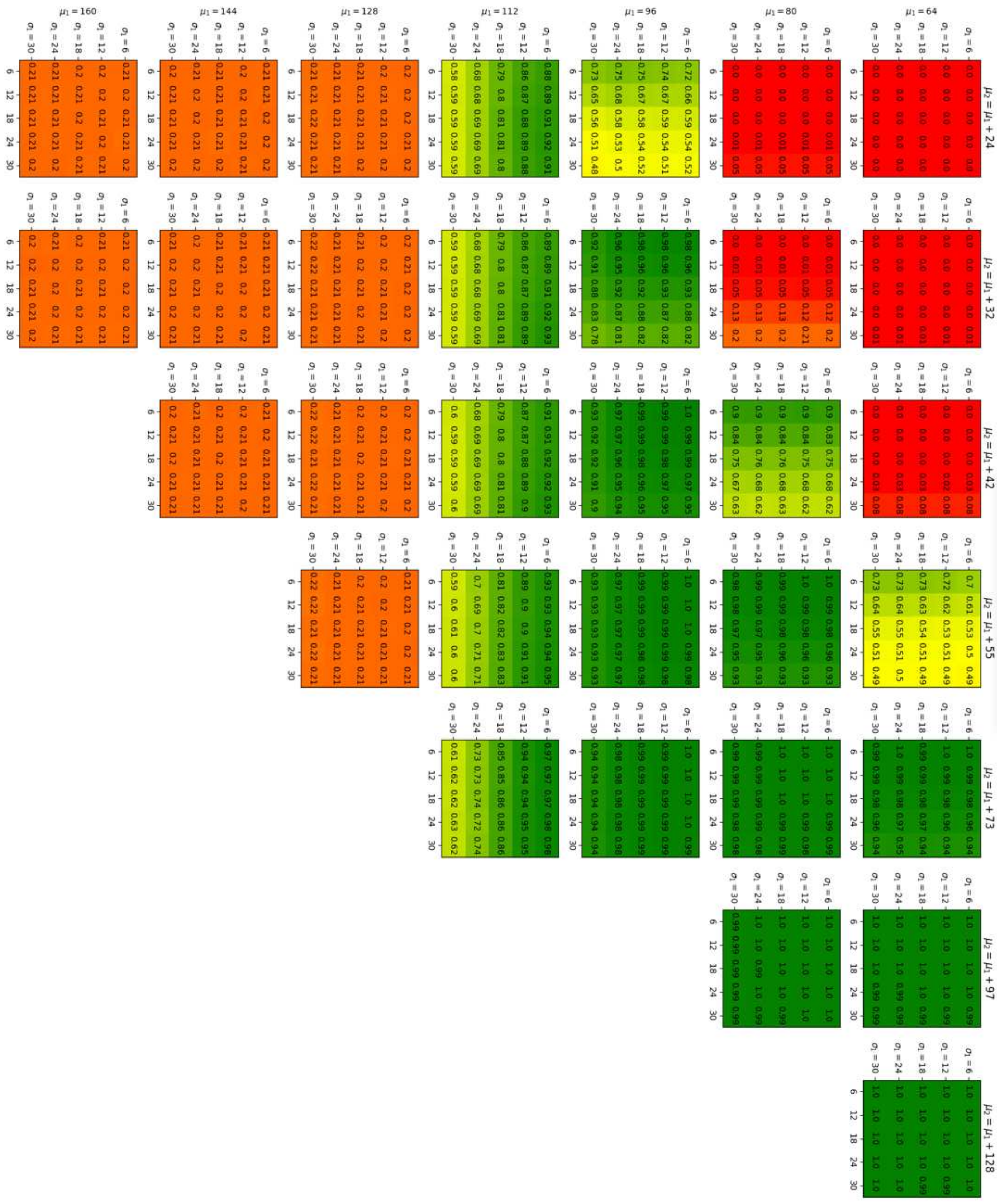


Figure 5.16: Results of the 1-800 experiment before BN adaptation. Note that indexes are shifted, $(0, 1) \rightarrow (1, 2)$. The x-axis in the subgraphs is σ_2.

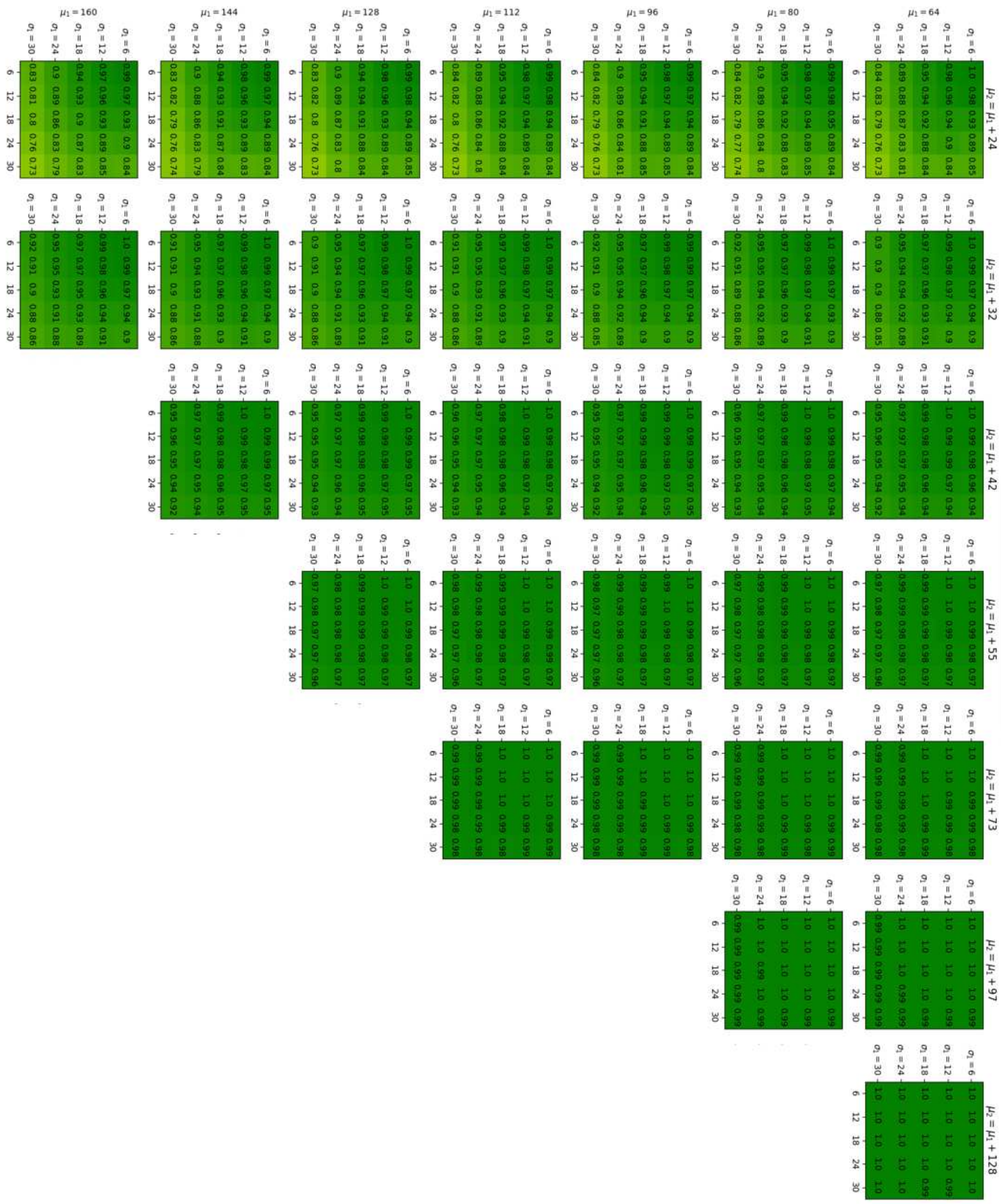


Figure 5.17: Results of the 1-800 experiment after BN adaptation. Note that indexes are shifted, $(0, 1) \to (1, 2)$. The x-axis in the subgraphs is σ_2.

5.4.1 Trying to predict BN adaptation performance: original parameters

As we are trying to predict the BN adaptation performance, the first intuitive test is to see if we can find a correlation between the original parameters $(\mu_0, \sigma_0, \mu_1, \sigma_1)$ and the IoU of the adapted models. To be exact, we are dealing with 8 parameters in total: the 4 parameters of the source $(\mu_{0s}, \sigma_{0s}, \mu_{1s}, \sigma_{1s})$ and the 4 parameters of the target $(\mu_{0t}, \sigma_{0t}, \mu_{1t}, \sigma_{1t})$, but since we have a single source dataset we will be only using the target parameters.

We tested different distances between the distributions $N(\mu_{0t}, \sigma_{0t})$ and $N(\mu_{1t}, \sigma_{1t})$, specifically the **overlap coefficient**[20], the **Wasserstein distance**, the **source-normalized Wasserstein distance**, and the **target-normalized Wasserstein distance** (refer to Section 2.5):

- $OVL(N(\mu_{0t}, \sigma_{0t}), N(\mu_{1t}, \sigma_{1t}))$ (Figure 5.18)
- $W_2^2(N(\mu_{0t}, \sigma_{0t}), N(\mu_{1t}, \sigma_{1t}))$ (Figure 5.19)
- ${}^{sn}W_2^2(N(\mu_{0t}, \sigma_{0t}), N(\mu_{1t}, \sigma_{1t}))$ (see Figure 5.20)
- ${}^{tn}W_2^2(N(\mu_{0t}, \sigma_{0t}), N(\mu_{1t}, \sigma_{1t}))$ (see Figure 5.21)

Ultimately, the most representative one is the coefficient of overlapping OVL, but it shows regardless a lot of outliers from the general trend.

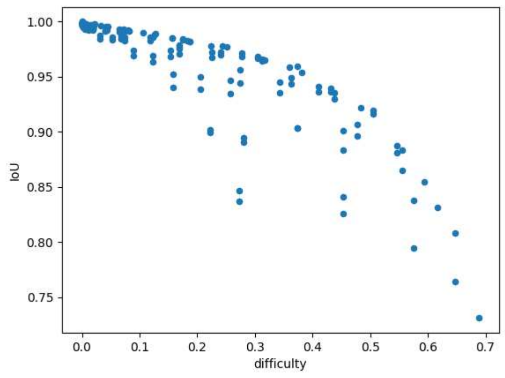


Figure 5.18: adapted IoU vs coefficient of overlapping $OVL(N(\mu_{0t}, \sigma_{0t}), N(\mu_{1t}, \sigma_{1t}))$

These graphs gives us an idea of the difficulty of this task, tricky to achieve even with the knowledge of the noise parameters. Anyways, since we want to predict BN adaptation performance in a general way, we will below see if the latent spaces analysis can lead to some significant result.

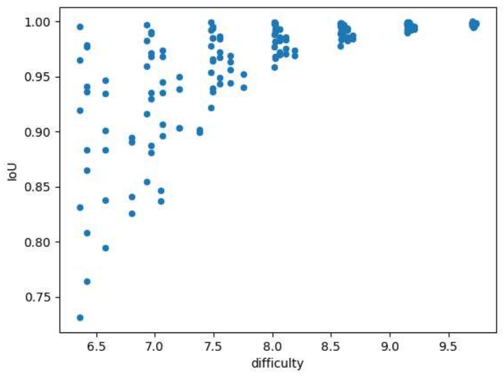


Figure 5.19: adapted IoU vs log Wasserstein $W_2^2(N(\mu_{0t}, \sigma_{0t}), N(\mu_{1t}, \sigma_{1t}))$

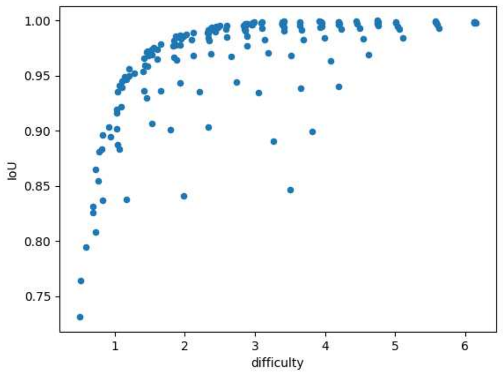


Figure 5.20: adapted IoU vs log source-normalized Wasserstein ${}^{sn}W_2^2(N(\mu_{0t}, \sigma_{0t}), N(\mu_{1t}, \sigma_{1t}))$

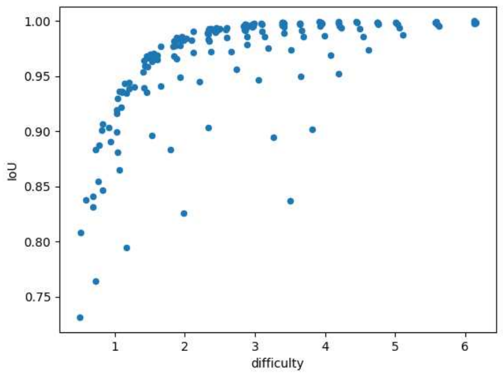


Figure 5.21: adapted IoU vs log target-normalized Wasserstein ${}^{tn}W_2^2(N(\mu_{0t}, \sigma_{0t}), N(\mu_{1t}, \sigma_{1t}))$

5.4.2 TRYING TO PREDICT BN ADAPTATION PERFORMANCE: LATENT SPACES

Here we discuss the graph of the BN adaptation performance vs the Wasserstein distance between the latent spaces. We show two graphs, the one relative to the log Wasserstein (see 5.22) and the one relative to the log target-normalized Wasserstein (see 5.24) as in [1]. Both graphs are relative to the 0-th latent space "ls = 0", as named in Figure 5.14.

log Wasserstein

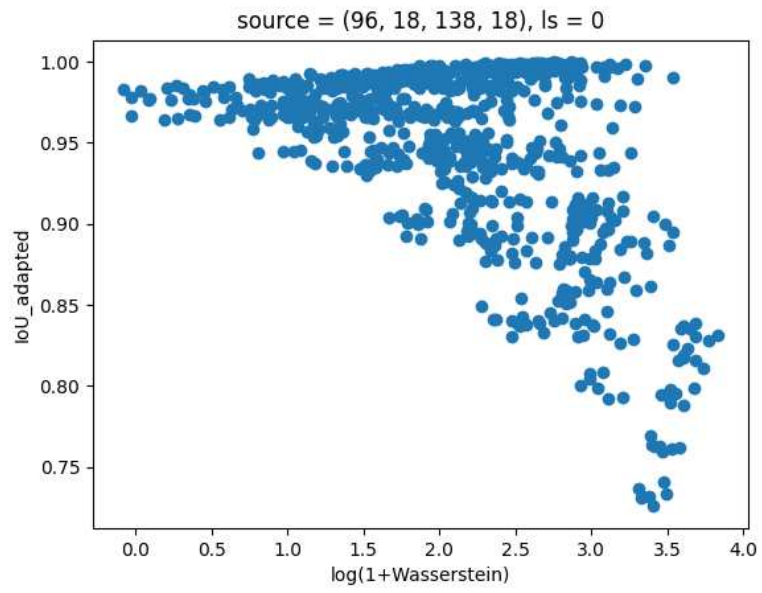


Figure 5.22: This graph shows the log Wasserstein vs the IoU of the BN-adapted U-Nets.

There is unfortunately no sign of a function-like behaviour between the two quantities as the one seen in Figure 3.2, which is anyways very unlikely to find. On the other hand, we can see clearly the possibility to set a lower bound on the IoU in function of the log Wasserstein. This is great news since it gives us a criterion to assess when BN adaptation works: **the smaller the Wasserstein distance, the better the adaptation**. Let us not consider the target-normalized Wasserstein, which was the one used in [1].

log target-normalized Wasserstein

This graph offers the opportunity for an even stricter lower bound!

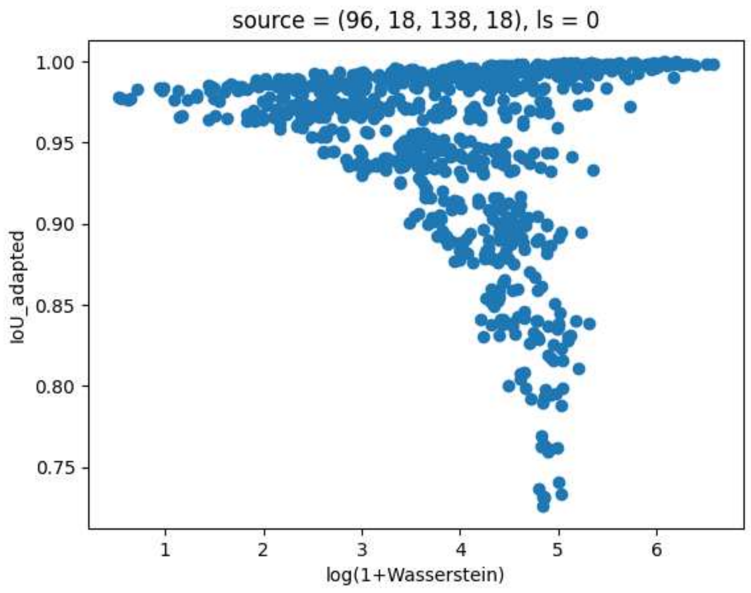


Figure 5.23: This graph shows the log target-normalized-Wasserstein vs the IoU of the BN-adapted U-Nets.

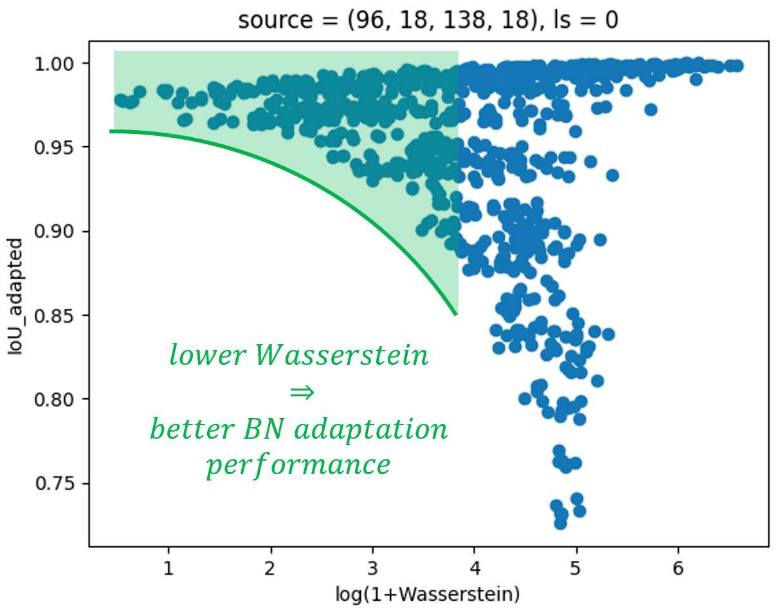


Figure 5.24: A lower Wasserstein is indicator of a better BN adaptation performance.

Note that even if there seems to be potential for an additional upper bound for values with $\log(1 + W) \geq 5$, we assume it is just due to the scarcity of data in that region.

This results opens a lot of paths to follow for future research. Specifically:

- We have now established a pathway to determine a sufficient condition for BN adaptation to work, which is already a significant achievement. Future research could focus on identifying the necessary conditions for BN adaptation to be effective. This would provide a comprehensive, unsupervised method for assessing the quality of BN adaptation.
- plotting other latent spaces, of course only if the Wasserstein computation is feasible in such latent spaces as shown in Table 5.1, and checking if the lower bound can be improved, or if any new pattern emerges.
- testing if the lower bound also holds on new toy datasets, or on real datasets.

The latter is currently a personal work in progress.

5.4.3 The 1-800 experiment: triangles and circles

The 1-800 experiment presented in the previous Section is the most relevant and complete one. Here we present one last experiment, of which we have the IoU but unfortunately not have a latent space analysis itself since it was completed towards the end of the internship. It is similar to the 1-800 one but with the added complexity of recognizing shapes, it was made to check if results would drastically change or stay close to the original 1-800. Results are shown in Figure 5.26 and 5.27.

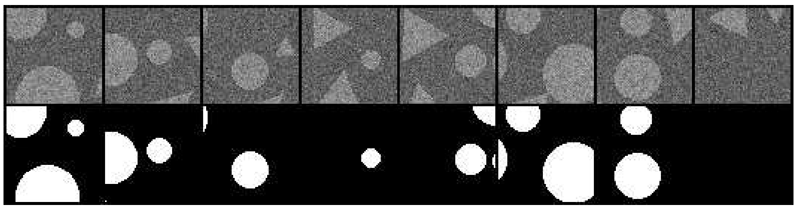


Figure 5.25: Some samples from the triangles and circles experiment

Details: 5000 source images, 1000 test images, batch size = 1, 5 epochs, early stopping with patience 5 and $\delta = 0$, percentage of validation set = 10%, learning rate = 10^{-5}, no image scaling, RMSprop optimizer (weight decay 0, momentum 0.999, gradient clipping 1.0) with ReduceLROnPlateau scheduler. Loss function = CrossEntropyLoss + DiceLoss.

Observations:

- This experiment was redone with different shapes and masks and background, results were analogous so we only showcase this one as representative.
- BN adaptation consistently works, despite the task having the added complexity of shape recognition on top of shape detection. Regardless, some exceptions are present.

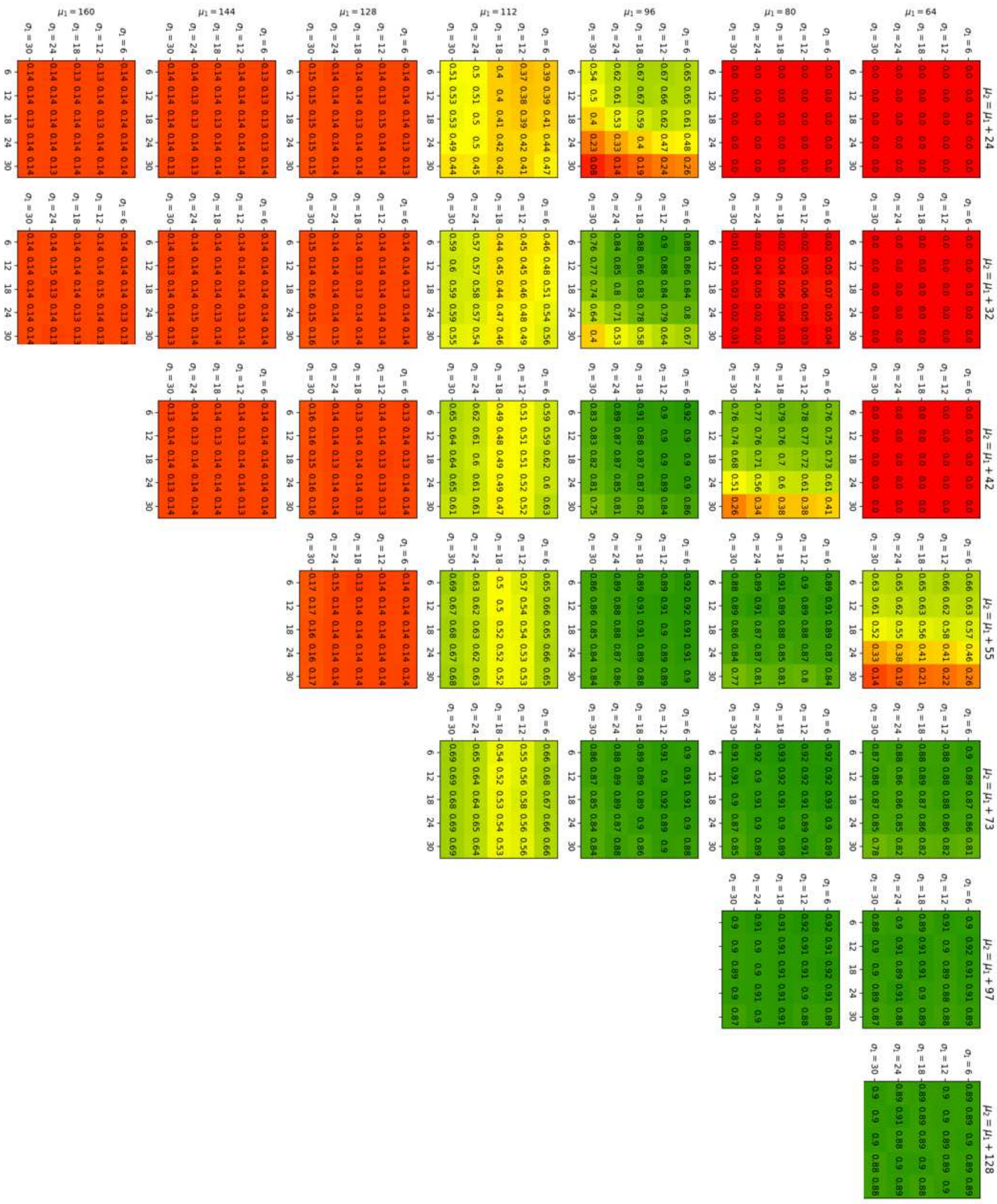


Figure 5.26: Results of the 1-800 experiment triangles and circles, before BN adaptation. Note that indexes are shifted, $(0, 1) \rightarrow (1, 2)$. The x-axis in the subgraphs is σ_2.

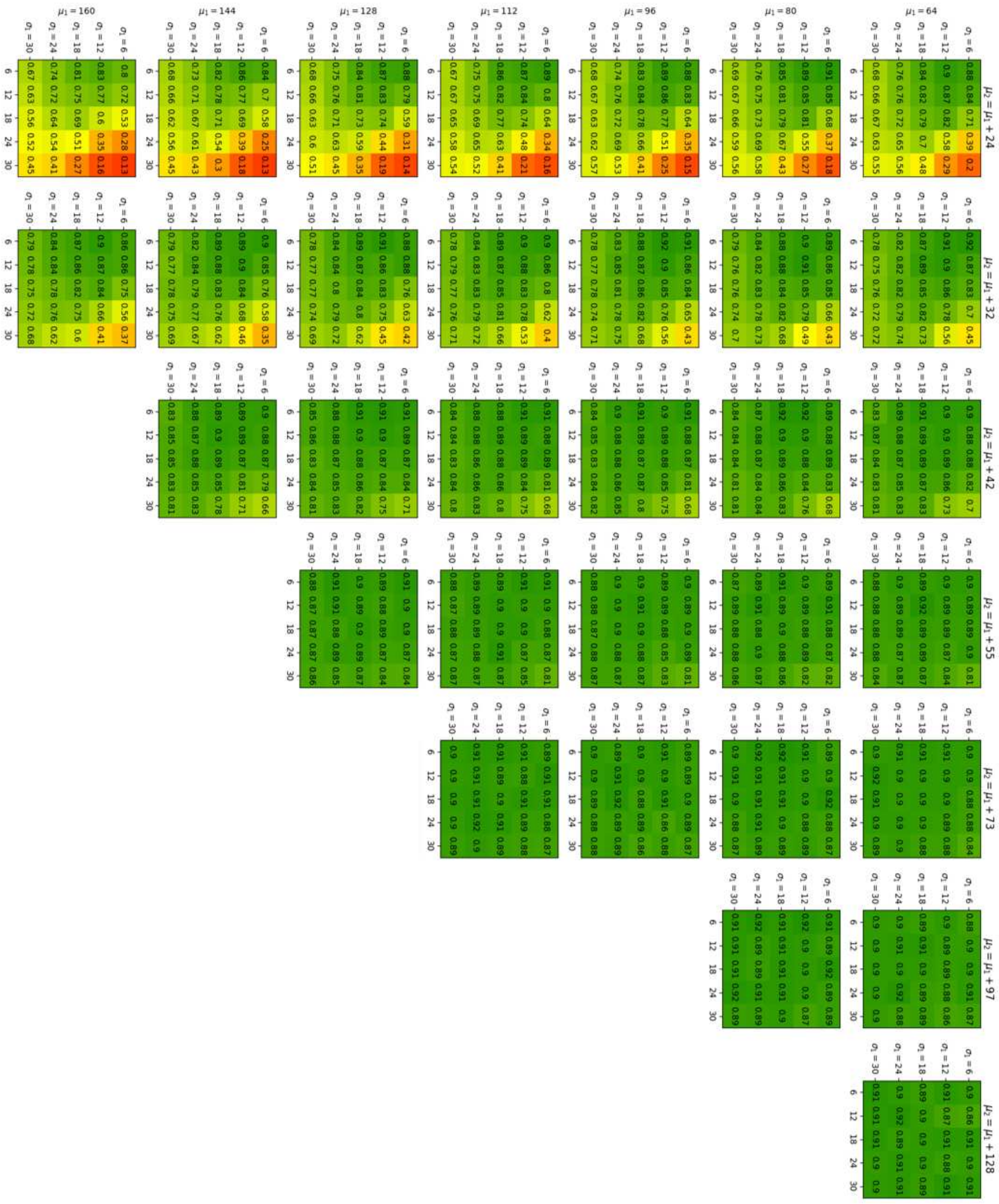


Figure 5.27: Results of the 1-800 experiment triangles and circles, after BN adaptation. Note that indexes are shifted, $(0, 1) \rightarrow (1, 2)$. The x-axis in the subgraphs is σ_2.

6 Conclusion

In conclusion, the task of predicting the BatchNorm adaptation efficacy on new target datasets in an unsupervised way revealed to be a tricky task. Regardless my Internship brought new discoveries and birthed new ideas for future research work on the field. My contributions regard both the practical, coding side and the theoretical side. Coding-wise, I achieved the following:

- A **simple and fast dataset generator** was developed to simulate datasets with mithocondria shaped masks. The images can be crafted from the masks with countless different techniques, allowing for a very general landscape of distribution shift scenarios.
- The **UNet code** was rewritten from the original code to optimize efficiency (removing useless lines of code that would slow everything down) and readability (all hyperparameters are cleanly organized in a single file instead of being all around the code).
- Developed utility functions for image segmentation and **a stable versions of the Wasserstein formula**.

All the codes mentioned are publicly available on my GitHub [*]. When it comes to the research itself, these were the key findings from my Internship:

- Upon thousands of BatchNorm adaptation trials, the cases in which it improved the performance vastly outnumber the cases in which it decreased it. So whenever there is an instance of domain shift, until further discoveries are done in the field, **it is generally better to apply BatchNorm adaptation for better results**.

[*] https://github.com/MarcoFurlan99/Marco_code_final

- As an example of the point above, as we saw in Experiment #1, **BatchNorm can correct irregularities in the training**: if the algorithm converges to a different local minima, which does not adapt properly to new target datasets, applying BN adaptation can fix it, making it more homogeneous with respect to the neighbourhood of similar dataset distributions.

- In some contexts applying BatchNorm adaptation works amazingly, as in the *brighness shift* example.

- **The Wasserstein distance computed on latent spaces provides a lower bound on the BN adaptation performance**. Naturally, this result requires further empirical investigation for clear assessment. However, the premises established by my experiments all convey towards this line of thought, and the underlying intuition is robust.

- If further investigations on the matter are carried, **The Wasserstein distance should always be computed on appropriate latent spaces** as shown in Table 5.1. If the latent space has small width and height but high depth (that is, features), the computation is not reliable.

References

[1] A. Stenger, L. Verdenne, P. Schultz, S. Faisan, E. Baudrier, and B. Naegel, "Fast and interpretable unsupervised domain adaptation for fib-sem cell segmentation," *ISBI*, 2022.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[3] K. P.Murphy, *Probabilistic Machine Learning - Advanced Topics*. The MIT Press, 2023, pp.727-743.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[5] Batchnorm2d - pytorch. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

[6] G. Du, X. Cao, J. Liang, X. Chen, and Y. Zhan, "Medical image segmentation based on u-net: A review," *Journal of Imaging Science and Technology*, vol. 64, 2020.

[7] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou, "Revisiting batch normalization for practical domain adaptation," *arXiv preprint arXiv:1603.04779*, 2016.

[8] Wasserstein metric as optimal cost of mass transportation. [Online]. Available: https://en.wikipedia.org/wiki/Wasserstein_metric#Intuition_and_connection_to_optimal_transport

[9] R. A. Horn and C. R. Johnson, *Matrix analysis (2nd ed.), p. 439, Theorem 7.2.6*. Cambridge university press, 2013.

[10] C. R. Givens and R. M. Shortt, "A class of wasserstein metrics for probability distributions," p. 9 (Corollary), 1984.

[11] Fréchet inception distance. [Online]. Available: https://en.wikipedia.org/wiki/Fr%C3%A9chet_inception_distance

[12] S. Schneider, E. Rusak, L. Eck, O. Bringmann, W. Brendel, and M. Bethge, “Improving robustness against common corruptions by covariate shift adaptation, p.15,” *Advances in neural information processing systems*, vol. 33, pp. 11 539–11 551, 2020.

[13] S. Schneider, E. Rusak, L. Eck, O. Bringmann, M. Bethge, and W. Brendel, “Improving robustness against common corruptions by covariate shift adaptation,” p. 15, 2020.

[14] pvigier. A fast and simple perlin noise generator using numpy. [Online]. Available: https://github.com/pvigier/perlin-numpy

[15] Perlin noise - wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Perlin_noise

[16] Pytorch-unet. [Online]. Available: https://github.com/milesial/Pytorch-UNet

[17] R. Novak, L. Xiao, J. Lee, Y. Bahri, G. Yang, J. Hron, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, “Bayesian deep convolutional networks with many channels are gaussian processes,” pp. 5, refer to paragraph *”Our work concerns proving ... ”*, 2020.

[18] pytorch fid. [Online]. Available: https://pytorch.org/ignite/generated/ignite.metrics.FID.html#fid

[19] pytorch function - calculate_frechet_distance. [Online]. Available: https://github.com/mseitzer/pytorch-fid/blob/master/src/pytorch_fid/fid_score.py#L179

[20] M. Vijaymeena and K. Kavitha, “A survey on similarity measures in text mining,” *Machine Learning and Applications: An International Journal*, vol. 3, no. 2, pp. 19–28, 2016.

Acknowledgments

First and foremost, I want to express my gratitude to all professors and PhDs who guided me throughout the Master's Degree in Data Science at the University of Padua. Their lectures and teachings will be fundamental throughout my future career. On this topic, I extend special thanks to Professor *Lamberto Ballan* for his Vision and Cognitive Systems course, which provided crucial concepts for my internship, and for his supervision of the thesis itself.

This thesis is not solely my achievement, but the result of a collective teamwork: I therefore extend my sincere thanks to Doctorant *Alexandre Stenger*, Professor *Étienne Baudrier*, and Professor *Sylvain Faisan* for their expert help and supervision throughout my internship. Furthermore, I thank all the interns, PhDs, professors, and researchers that have followed me and helped me during my six months at the ICube Laboratories.

I want to express my gratitude to all my colleagues and friends who accompanied me throughout the Data Science Masters program, providing invaluable support during the hours of study, coding, and group projects that we did together.

A warm thanks to my closer relatives - aunts, uncles and cousins - and to my close friends for being there for me and guiding me throughout my life.

Lastly and most importantly, a heartfelt thanks to my brother, my mother and my father, for supporting me through all these years. Their love and belief in me have been the driving force behind every achievement along this journey.