Use GANs to deform 3D bodies.

Joseph ABBOUD
Vincent NAAYEM
Zad ABI FADEL

***Abstract*—This report is written in the context of the second project for the Machine Learning course for masters at EPFL (CS433). The work done is part of the ML4Science project, in collaboration with the Laboratory of the Physics of Biological Systems (LPBS) at EPFL [1], under the supervision of Prof. Sahand J. Rahi and Ms. Perrine Woringer.**

I. Introduction

Caenorhabditis elegans (C. elegans for short) is a worm known as a nematode which is $1mm$ long [2]. The project consists of tracking neurons of such worms. Since the worms are deforming objects, and since gathering a large amount of training data is very hard in the context of this experiment (Capturing images of worms and rendering them, to then pick the "clean" enough pictures), the use of a GAN [3] is one way to start with a single image, and deform it in order to obtain realistic images of worms which can then be used as training data for the tracking.

We have been given the code of a GAN that was already developed by the lab, but it was not working completely right and even though some results were positive, others were not. Our main task was to find out what was causing these negative results, and then maybe try to find a solution in order to fix them and make the GAN work as expected. The code can be found at https://github.com/Zad2/ML4. Instructions on how to make the code run can be found in the repository.

***Note 1*:** *Even though there are a lot of parts that were added by us, the GAN's main skeleton and the worm dataset are to be kept private since the lab has not yet finished with the study. We have taken Prof. Rahi's permission to share it with the correcting team.*

***Note 2*:** *We were three teams in total working on this project. So some of the relevant scenarios to try out have not been included in our study, because another group worked on them and it would be redundant to do it ourselves. Such cases will be specified later in the report*

II. Code & Data Familiarization

First and foremost, before getting into experimenting and trying out ideas, one must understand exactly the structure of the GAN that is analyzed, as well as the nature and distribution of the dataset that is being used. The input images are resized into 128x128 pixels per image.

Generator

The Generator possesses 17 layers, which go as follows:

1) conv2d_1: (in: 100 — out: 64 — stride: (1,1) — no padding)
2) batchnorm2d_1
3) leakyReLu_1
4) conv2d_2: (in: 64 — out: 32)
5) batchnorm2d_2
6) leakyReLu_2
7) conv2d_3: (in: 32 — out: 16)
8) batchnorm2d_3
9) leakyReLu_3
10) conv2d_4: (in: 16 — out: 8)
11) batchnorm2d_4
12) leakyReLu_4
13) conv2d_5: (in: 8 — out: 4)
14) batchnorm2d_5
15) leakyReLu_5
16) conv2d_6: (in: 4 — out: 2)
17) tanh

All leaky ReLu activation functions have negative slope 0.2 and all convolutional layers have kernel size (4x4), stride (2,2) and padding (1,1), unless stated otherwise.

The Generator, as its name states, takes as input a single image, and uses deformation fields to create new images resembling the training data. Its goal is to *"fool"* the discriminator defined below into thinking that generated *"fake"* images come from the real training data.

Disriminator

The Discrimnator possesses 16 layers, which are:

1) conv2d_1: (in: 1 — out: 4)
2) leakyReLu_1
3) conv2d_2: (in: 4 — out: 8)
4) batchnorm2d_1
5) leakyReLu_2
6) conv2d_3: (in: 8 — out: 16)
7) batchnorm2d_2
8) leakyReLu_3
9) conv2d_4: (in: 16 — out: 32)

10) batchnorm2d_3
11) leakyReLu_4
12) conv2d_5: (in: 32 — out: 64)
13) batchnorm2d_4
14) leakyReLu_5
15) conv2d_6: (in: 64 — out: 1 — stride: (1,1) — no padding)
16) sigmoid

The last layer is a sigmoid activation function, but it can be changed to tanh depending on the loss function used. We have found out that Mean Binary Cross Entropy Loss (BCE) is the most appropriate for the model. (see later). Like the Generator, all leaky ReLu activation functions have negative slope 0.2 and all convolutional layers have kernel size (4x4), stride (2,2) and padding (1,1), unless stated otherwise.

The goal of the Discriminator is to take as input an image and give a probability on whether that image comes from the training data (real). or was created by the Generator (fake). The GAN's ultimate goal is to reach a Discriminator accuracy of 50%, meaning that it cannot distinguish between a real and a fake image.

LQ dataset

The training data is composed of images representing neurons of C. elegans. As we can see in Figure 1, there are two main shapes composing the images: the blobs are the **soma** and the filaments are the **neurites**. A realistic deformation should take into account all elements no matter the brightness and the shape. Also, the GAN should not enter into mode collapse, that is when several deformed images are almost identical, despite coming from different noise samples.

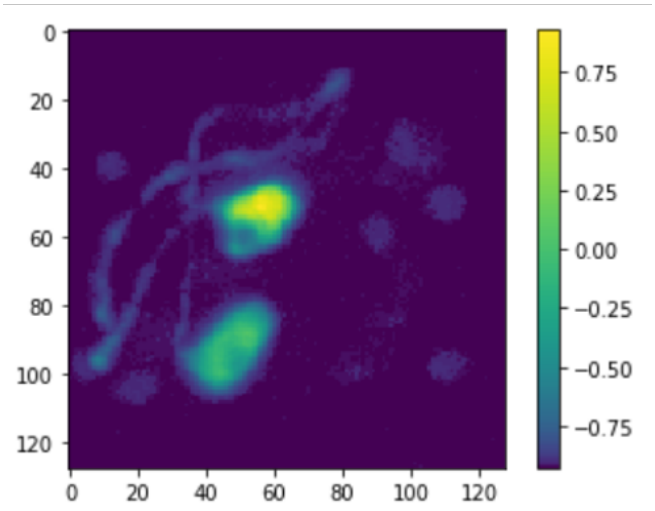

Figure 1. Visualization of neurons for C. elegans

Loss Functions

There were several potential loss functions that were considered when training the GAN:

- **Binary Cross Entropy Loss (BCE)**:
  $l(x, y) = mean(L) = mean(\{l_1, ..., l_N\}^T)$
  Such that
  $l_n = -w_n[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$
  A sigmoid function is used as last layer for the Discriminator in order to get predictions in $\{0, 1\}$
- **Hinge Loss**:
  $l(x, y) = mean(L) = mean(\{l_1, ..., l_N\}^T)$
  Such that
  $l_n = x_n$ if $y_n = 1$ and $l_n = max\{0, \Delta - x_n\}$ if $y_n = -1$
  A tanh function is used as last layer for the Discriminator in order to get predictions in $\{-1, 1\}$
- **Fréchet Inception Distance (FID)**:
  This function was investigated by another group and it has been concluded by us that even though it is useful in measuring the quality of generated samples, it did not have exactly a place in our work.

Moving forward, our examples only have used BCE since it yielded the best results with the current GAN architecture.

III. Methodology

***Note*:** We have conducted many experiments, so showing all the results will lead to a very noisy report. You can find our most relevant data on this drive [4]. It contains three folders, one for the pickled data, another containing the images and the parameters given to our code in order to generate them, and the third contains the results when we trained the GAN on those images.

First of all, since there are less than 600 worm images, we decided to inspect them manually and look for outliers and faulty images. Such images can be badly cropped or the result of a mistake from the camera (Figure 2).

After the cleanup, we ran the code with both initial and *"clean"* datasets, and as we see in _datasetfig:LQ_datasetFigure 3, we do not see a different accuracy from the Discriminator. In other words, the Generator realises that these images are bad and just ignores them when deforming the initial image. Also, we can see that with the initial worm dataset, D can spot a fake image 70% of the time, which is not bad, but still very far from 50%.

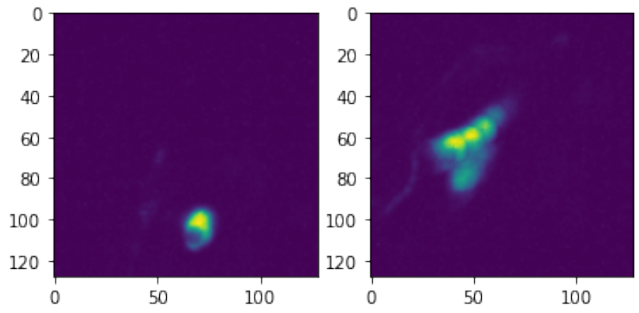

Figure 2. Two examples of a bad image

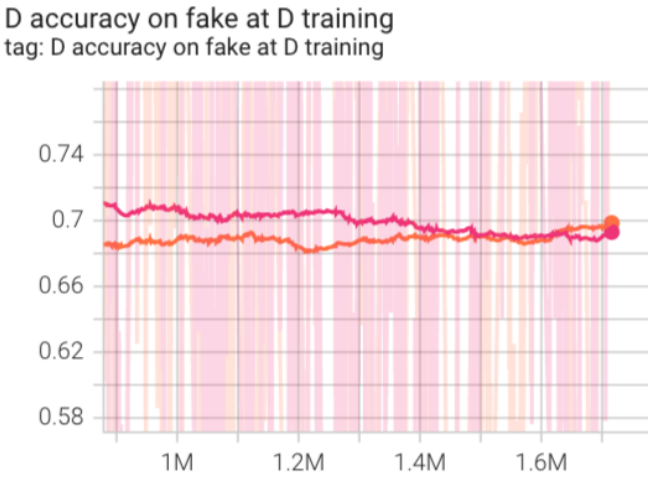

Figure 3. Initial runs

In order to understand what is going wrong with the initial run, we decided to go back to the basics. Instead of feeding the GAN complicated worm images, why not start from a more simple image, and see how the GAN performs. Our first idea was to create images containing two ellipses representing the soma, with different centers and sizes, without any noise (Figure 4). The difference in colors in our figures in this report should not be taken into account, since our code normalizes the images and uses the same color map.

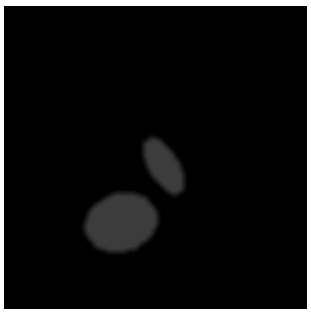

Figure 4. Two random ellipses

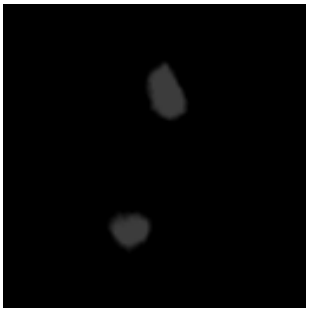

Figure 5. Deformed ellipses with fixed centers

Figure 6. Result of non-deformed and deformed fixed-center ellipses

The results were very bad: the discriminator's accuracy on fake images was around 86%. After this, we thought that it was maybe the fact that the ellipses were randomly placed in the image, since in the real dataset, the two main soma are placed in a specific position relative to each other as in Figure 1. The next step consisted on constructing synthetic images similar to the worms, where the only variation is the ellipses' rotation around their center. We then deformed this dataset with a manually coded deformation field (see Figure 5) and applied very light gaussian blur to not end up with rough edges.

The results were much better. As we can see in Figure 6, these are the results when the training data consists of fixed-center rotated ellipses and the same data when passed through a deformation field. We are close to 50% and we can say that the GAN's issue does not come from the deformation per se. It means that artifacts start appearing when other elements such as noise are included. It also could be due to the fact that while deforming soma is straightforward, deforming neurites can be a more difficult task.

This is why we added to our synthetic data an approximation of neurites, which was connecting the ellipses' centers by Bézier curves. After running the code

again, the Discriminator's accuracy was again very bad at nearly 100%, which could be due to the fact that these curve's edges, even though blurred, were still easily detectable by the Discriminator when they differ from the training data.

After this, our next logical experiment was to add noise to the synthetic data. We added three different types of noise: Gaussian with low, high and very high variance, Poisson and Speckle. In every case, the Generator entered into mode collapse, and the Discriminator's accuracy to detect fake images was at 100%. This was great in showing a major flaw in the GAN, **noise**. Since fake images are generated by passing through a deformation field, it looks like the Generator has a hard time differentiating between the clean data and the noise.

Going back to the worm dataset, we thought that since noise is a problem, why not blur the worm images and see what happens when feeding them to the GAN. We used gaussian blur with kernel sizes 3x3, 11x11, 51x51 and 127x127. This yielded two bad results: first of all, the Discriminator's accuracy did not get better. And secondly, the less bright parts of the worm in an image was considered as noise, and we ended up with a dataset resembling our synthetic one, with only two blobs.

Finally, our last experiment was to use the worm dataset as training data, but use a synthetic image as initial input. The thought behind that was that maybe the simplicity of the initial image might help the Generator in deforming it into the worms. We were wrong again, as the Generator entered into mode collapse.

IV. Potential Fixes

To conclude, we will try to propose different ways to try and improve the GAN, after looking at our experiments and seeing what works and what does not.

We think that different types of structures present in the images (soma of varying brightnesses, neurites) are making the deformation too complicated for the Generator. We propose the following solution:

Separate the worm dataset into three different ones. Each image is separated into an image containing only the two bright soma, another image containing all other soma, and a third image containing the neurites. We don't know if that is possible with computer vision tools since the goal behind the creation of the GAN is to generate a training dataset to segment the images into these different parts. To paraphrase Prof. Rahi, we would be using the solution to try and find the solution.

But since the number of datapoints is relatively low (less than 600 samples), the segmentation of might be possible to be done by hand.

After that, we can train a GAN for each of these three subdatasets, trying to generate more of the individual parts of the worm's neurones, to then merge them into one image by superposition. That solution might be practically unfeasible and is completely theoretical, but we think the problem comes from the diversity of segments in each image, and a strategy of divide and conquer might prove useful.

V. Acknowledgements

A special thanks to Prof. Sahand J. Rahi and Perrine Woringer in giving their time to show us the practical part of Machine Learning.

References

[1] "Laboratory of the physics of biological systems," https://www.epfl.ch/labs/lpbs/, 2021.

[2] W. Wood, "The nematode caenorhabditis elegans," 1988, p. 1. ISBN 978-0-87969-433-3.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," 2014, proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.

[4] "Project drive," https://drive.google.com/drive/u/1/folders/0AErO7RRT3WMmUk9PVA, 2021.