CBIO – Centre for Computational Biology

Université Paris-Dauphine

Mathématiques et Informatique de la Décision et des Organisations

In silico labeling for phase contrast microscopy

Supervisors :

Arthur Imbert
Thomas Walter

Maxence Philbert

M2 Intelligence Artificielle Systèmes Données

Novembre 2020

Contents

List of Figures

Acknowledgements

I would like to thank CBIO laboratory for welcoming and integrating me despite the difficult circumstances caused by the pandemic. I would particularly like to thank my supervisors Arthur Imbert, Thomas Walter and Florian Mueller for providing me great guidance and help throughout this internship. I am very grateful to Arthur for the time he devoted to me as well as his desire to help me learn and grow, especially during our many Friday "paper meetings". I also thank Thomas for his trust, his precious advice and his final review on this master's thesis.

1 Introduction

1.1 In Silico Labeling (ISL)

In Silico Labeling (ISL) is a deep-learning approach developed in 2018 aiming at predicting fluorescence microscopy images from non-invasive imaging techniques, such as transmitted light microscopy. In this introduction section, we present the relevant microscopy techniques, namely transmitted light and fluorescent microscopy. We then present the idea of ISL and finish by outlining the objectives of this internship.

1.1.1 Microscopy imaging : transmitted light and fluorescence

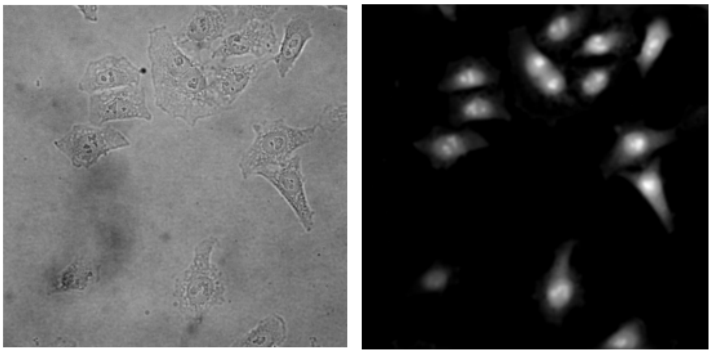
Light microscopy is one of the most well-known and most widely used research tools in biology to access information from various biological scales, across time and space. However, visualizing cellular structures can be challenging. Transmitted light microscopy is a non-invasive microscopy technique with which we can image the overall morphology of cells and cellular structures. Different variants have been defined to improve the notoriously low contrast. However, observing specific proteins or cellular structures in greater detail requires the use of different techniques. In particular, fluorescence labeling allows to specifically label macromolecular structures and the subcellular components where they localize. Both methods have their advantages and disadvantages, and can be complementary.

Transmitted light microscopy

Transmitted light microscopy is the most traditional microscopy technique. The light source and the detection objective of the microscope are placed on opposite sides of the sample. As the light passes through the sample, a part is absorbed to varying degrees by the structures in the sample. These differences in absorption are then used to produce an image. Usually the light is passed through a condenser to focus it on the sample in order to get very high illumination. After the light passes through the sample, the light goes through the microscope objective lens and then to the oculars where the enlarged image is viewed.

The simplest microscopic technique requiring a transmitted light path is Brightfield. However, because most cells are thin and transparent, they do not absorb much light and are difficult to see. Therefore, it is common to add optics that allow to visualize the phase shift of light induced by the sample. The two most commonly used techniques to visualize this phase shift are phase contrast and differential interference contrast (DIC). Brightfield itself is usually sufficient to see the general outlines of cells, but phase contrast or DIC is necessary to achieve detailed, high-contrast transmitted light images.

- Phase-contrast microscopy is an optical microscopy technique that converts phase shifts in light passing through a transparent sample, to brightness changes in the image. Phase shifts themselves are invisible, but become visible when shown as brightness variations. It allows to reveal many cellular structures that would be invisible with a Brightfield microscope. Earlier microscopists were able to observe these structures thanks to staining, but this required additional preparation usually implying cell death. The phase-contrast microscope made it possible for biologists to study living cells and in particular dynamic biological processes, such as cell division and migration. Phase-contrast microscopy is one of the few methods available to quantify cellular structure and components without using fluorescence.



(a) Brightfield image (b) Phase contrast image

Figure 1: Brightfield and phase contrast images examples

- Differential interference contrast (DIC) is a microscopy technique that intensifies contrast in images of samples which have little or no contrast when viewed using brightfield microscopy. DIC works on the principle of interferometry to gain information about the optical path length of the sample, to see otherwise invisible features. A relatively complex optical system produces an image with the object appearing black to white on a grey background. This image is similar to that obtained by phase contrast microscopy but without the bright diffraction halo, and gives a pseudo–three-dimensional (3D) shaded appearance to cells.

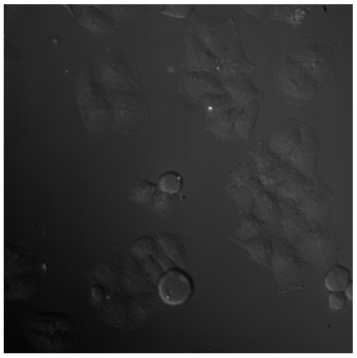
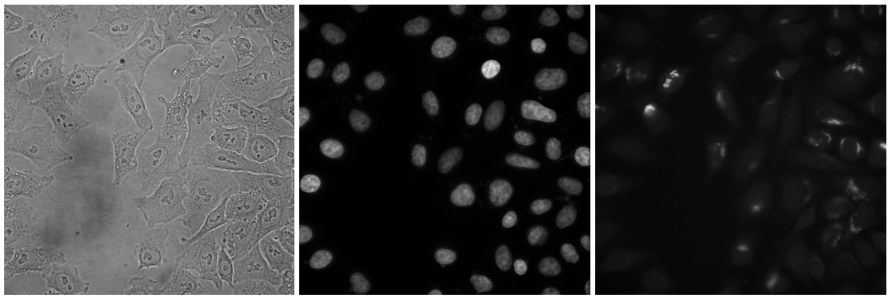


Figure 2: Example of a DIC microscopy image

Fluorescence microscopy

Fluorescence microscopy is the most widely used technique in bioimaging today. This microscopy technique, largely developed in the nineties, allows the imaging of structures of interest by specific fluorescence labeling. Specifically, a particular protein of interest is labeled with a fluorescent molecule. After excitation, we can collect the light emitted by the fluorophores attached to the protein of interest by filtering out all non-specific wavelengths. This allows to acquire images with very good signal-to-noise ratio, where we observe exclusively the spatial distribution of the protein of interest. While this specificity is the major advantage of fluorescence microscopy, it also comes at a price: we are blind to all components of the cell that are not fluorescently labeled. Moreover, it requires advanced instrumentation and time-consuming sample preparation, and due to spectrum overlap, the number of simultaneous fluorescent tags is restricted. We can only use three to four fluorescent labels at the same time, and consequently, we are limited to observe only few proteins and structures at the same time.



(a) Brightfield image (b) Dapi image (c) Golgi image

Figure 3: Examples of fluorescence labeling. In (a), we observe the outlines of several cells in a brightfield microscopy image. We can observe in (b) the associated fluorescence image that labels the nucleus of each cell and in (c) the associated fluorescence image that labels the Golgi apparatus within each cell.

Transmitted light and fluorescence microscopy both have their advantages and disadvantages. There exist many variants of transmitted light microscopy technique, but they all have in common that they do not make use of any fluorescent label. Unlike fluorescence microscopy, they allow to collect light from all components of the sample that interfere with the transmitted light. Moreover, transmitted light microscopy is relatively low-cost, and is label-free. However, although valuable information about cellular organization is apparent, the images typically have low contrast and lack the clear contrast of fluorescence labeling.

A method combining the clarity of fluorescence microscopy with the simplicity and modest cost of transmitted light microscopy would present a groundbreaking tool for biological research. This is what originated the development of In Silico Labeling.

1.1.2 In Silico Labeling challenge

The obvious complementarity of the two microscopy approaches presented in the previous section has motivated computational researchers to develop “In silico labeling” (ISL). ISL is a deep-learning approach that aims at predicting fluorescence microscopy images from transmitted-light images of unlabeled biological samples. ISL relies on the hypothesis that transmitted light images contain much more information than what can be seen by eye. We therefore may be able to predict their associated fluorescence images without actually acquiring them. The goal is to develop a model that is able to find and predict features in unlabeled images that normally only become visible with invasive fluorescence labeling. Because prediction happens ”in silico”, the method is consistent and is not limited by spectral overlap of fluorophores or experimental issues. ISL allows to generate biological measurements that would otherwise be complicated or impossible to acquire.

In silico labeling has been proposed in 2018 as a technique, and has led to several interesting papers, that all provide proof-of-concept studies that show that for a number of proteins this prediction is very close to the original fluorescent images (Christiansen et al. 2018 [1], Weigert et al. 2018 [2]; Ounkomol et al. 2018 [3]; Belthangady et al. 2019 [4]). In their paper, Christiansen et al. [1] first acquire both fluorescence microscopy images with up to 3 channels (3 different fluorescent labels) and the corresponding label-free image by transmitted light microscopy (for example phase contrast or brightfield images). Then, they design a neural network capable of predicting the fluorescent channels from the transmitted light images. This neural network can then be applied to transmitted light images, for which no fluorescence counterpart has been acquired.

However, the method has not yet been used in a real-world biological application, where the objective is not simply to predict a fluorescent image, but to use it in order to answer biological questions. Also, it is not yet very clear which proteins can in principle be predicted and which are not suitable for this kind of approach. Finally, In Silico Labeling method can be more than simply predicting a fluorescent image : it can be very useful for solving other biological tasks such as cell or nuclei segmentation, using transfer learning. Here, we propose to tackle some of these exciting questions.

1.2 Internship tasks

1.2.1 Nuclei fluorescence images prediction

The first task consisted in performing In silico labeling for nuclei images, more specifically in predicting fluorescence images that label the cell nuclei, from transmitted light images. This has been proved to be feasible previously, so it was a good start to explore different settings of the In Silico Labeling method and to become familiar with the data. This part consisted in studying state of the art ISL methods and applying them to our data to see if it could easily predict our fluorescence nuclei images. For this task, we worked on two different datasets : one that used Brightfield and phase contrast as transmitted images to predict dapi images, and the second one with DIC images to predict Hoechst channel. In both DAPI and Hoechst images cell nuclei are fluorescently labeled.

1.2.2 Cell cycle fluorescence prediction

While ISL has been applied to different biological tasks such as predicting cell nuclei or various subcellular structures, it hasn't been shown yet that it is possible to use ISL to predict cell cycle phases. One of this internship challenges was to tackle this question and try to use In Silico Labeling for predicting cell cycle stages. Indeed, cells evolve in a periodic sequence of different phases, called G1, S, G2 and M. G1 and G2 are ground states where the cell grows, S is the replication phase, where the DNA is replicated and M is the very short phase cell division phase. During each phase of the cell cycle, specific proteins are expressed, and cellular structures undergo important changes (e.g. DNA replication, chromosome condensation and segregatation, duplication of centrosomes, etc.). For this reason, cells within different cell cycle phases really can show very different behaviour, as a different transcriptional program is activated. This has many implications regarding the interpretability of experimental results in biology. For instance, we could study how the cell cycle influences certain drug treatments and thereby understand mechanisms of drug resistance.

In our dataset, we used the well-known FUCCI markers which are differentially expressed in different cell cycle phases: cells in G1 are labeled in red (Cy3) and cells in either S or G2 are labeled in green (GFP). The goal here was to predict these fluorescence images from DIC images, in order afterwards to be able to tell which stage of the cell cycle a cell belongs to, from its transmitted light image.

While the changes during the cell cycle are certainly dramatic at a molecular level, it was not clear whether such a prediction was actually feasible for most of the phases. Indeed, we know that one of the requirements of in silico labeling is that the predicted protein leaves a fingerprint in the transmitted light image, which is not clear for the phases G1, S and G2.

1.2.3 Application of In Silico Labeling to segmentation tasks

Image segmentation is certainly the most successful application of deep learning in the field of bioimaging. Indeed, in bioimaging, we are often confronted to complex segmentation tasks that were not solvable with traditional methods a few years ago. One caveat however is that deep learning usually requires a lot of annotated data. Many strategies have been developed in order to obtain larger annotated data sets, including citizen science and gamification, image simulation and weak supervision. Here, we would like to propose a conceptually different approach, where we use smart experimental setup in combination with in silico labeling to dramatically reduce the size of a training set without sacrificing performance.

The idea we would like to investigate here is to pretrain a segmentation network on the prediction of fluorescence microscopy data, where the objects we ultimately want to segment are fluorescently labeled. We hypothesize that this task is similar to the segmentation task and that for this reason we will only need very few annotated images to fine-tune the pre-trained network. We will show this for the example of nuclei segmentation. We will show that In Silico Labeling provides us with an excellent "pretext task" for nuclei segmentation that has really the potential to replace tedious image annotation to a very large extent.

2 State of the Art

Recent developments in neural networks and deep learning approaches have greatly improved the performance of many computer vision tasks. Throughout this section, we present how Convolutional neural networks have become a groundbreaking tool for many tasks such as image classification, object detection or segmentation. We also introduce Generative Adversarial Networks and develop their exciting adversarial framework, while discussing some of their limitations. Finally, we define transfer learning and present some of its many advantages.

2.1 Convolutional neural networks (CNN)

In this section, we define Convolutional neural networks (CNN), a class of artificial neural networks that has become dominant in various computer vision tasks and has attracted interest across many domains. CNN allow to extract essential features from images and to solve specific tasks by using multiple building blocks such as convolution layers, pooling layers and fully connected layers. Not very transparent, we also try to get an interpretation of these CNN models by studying their feature visualization. Finally, we give examples of CNN useful applications and present some state of the art CNN models.

2.1.1 Feature extraction : convolution block

A Convolutional neural network (CNN) is composed of a convolution block which consists in extracting features from the input image. The convolution block is composed of convolutional layers and pooling layers.

Convolutional layers

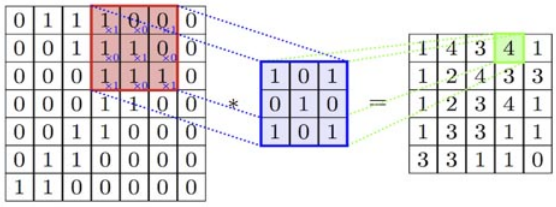


Figure 4: Application of a filter

Filters, also named kernels, are images composed of weights that depict a specific type of feature. Applying a filter to an input image enables to detect if the image contains a particular feature or not. For example, a filter can be a line or curve detector. When we talk about "applying a filter to an image", we are talking about a convolution mechanism. Here, the convolution is a linear operation that consists in computing the scalar product between the filter and a filter-sized patch of the input image. More specifically, it consists in an element-wise multiplication of each of their elements, which is then summed, resulting in a single value as output.

The filter is usually smaller than the input image so we repeat this operation multiple times in order to apply the filter to each filter-sized patch of the input data, left to right and top to bottom. The way you shift the filter within the image actually depends on the "stride" you choose, notion we will discuss later. This allows the same filter to "see" the input image at different points and therefore to detect the feature anywhere in the global image. In the end, after passing the filter through the entire image, we obtain an array of output values that represents a filtering of the input, commonly

called "feature map". Once a feature map is created, we can pass each value of the feature map through a nonlinear function, such as a ReLU, like we do for the outputs of a fully connected layer.

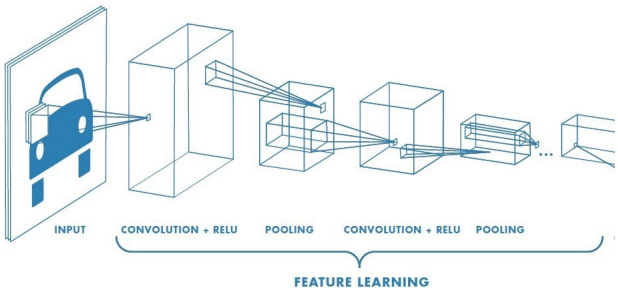


Figure 5: Convolution neural network : convolution block

A convolutional neural network is a neural network composed of convolutional layers. In convolutional layers, filters are applied to input data and the values of these filters are weights to be learned during the training of the network. More precisely, the network updates its filters weights thanks to backpropagation in order to minimize a loss that corresponds to the specific task it is being trained to solve. Therefore, the types of features the network learns to extract from the input depend on the task. For example, if we are trying to classify images as dogs or cats, a type of useful features that the network can extract is the shape of the ears (pointed ears for cats), which can serve as a discriminatory criterion to identify the animal in the image.

Usually, convolutional layers learn multiple filters at the same time for a given input, in order to extract features in different ways during training. This enables to learn features that are specific to the training data. In the case of images with multiple channels (for example color images RGB), the filter that we apply must have the same number of channels as the input, often called "depth". The process is basically the same : if we apply a three-dimensional filter to a three-dimensional image, each channel of the filter has its own weights and is applied to the corresponding image channel. The three feature maps obtained are summed in order to output a single and final feature map. Let's consider an input that has L channels and you want your convolutional layer to output K feature maps (so an output with K channels). Therefore by applying K filters of size $(N \times M)$, you actually have to learn the weights of K filters, each of size $(N \times M \times L)$. Taking in account a bias term, this gives an overall of $(N \times M \times L + 1) \times K$ parameters.

Convolutional neural networks are composed of sequential convolutional layers. Therefore, like fully connected layers in neural networks, convolutional layers are not only applied to input data, they can also be applied to the output of other layers. Filters learn to extract different types of features, depending on the depth of the network. Filters from the first layers will learn to extract low-level features such as lines and curves, whereas filters in deeper layers will be able to extract more complex features such as specific objects from the training data. We will discuss and illustrate this in more detail in section 2.1.2.

In a convolution neural network, each feature map unit in a hidden layer is only connected to a small number of units in the previous layer, instead of being connected to all of them like in fully connected networks. This allows to reduce drastically the number of parameters involved, which is very useful when we are dealing with high-dimensional inputs such as images. Specifically, the value of each unit depends on a particular subset of the previous feature map, that depends itself on a subset of the input image. Therefore, units from deep layers tend to cover a higher portion of the input image. The receptive field of a unit designates the local region of the input image that is seen

and responded to by the unit in question. The larger its receptive field is, the more a unit can encode contextual information in the input image.

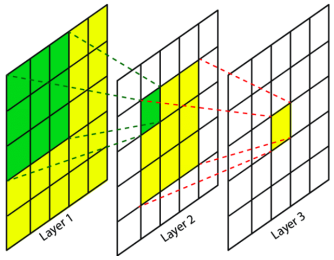


Figure 6: Receptive field

Stride, padding and pooling layers

As explained earlier, in a convolutional layer, when we apply a filter to an image, it is actually applied to multiple filter-sized patches in the image. Stride is a parameter of the convolutional layer that allows to control the amount of movement of the filter over the image, horizontally and vertically. A stride set to (1, 1) moves the filter across the image from left to right with a one-pixel column change and from top to bottom with a one-pixel row change. A stride set to (2, 2) moves the image two pixels at a time, and so on... Naturally, changing the stride affects the size of the resulting feature map. For example, an image of size 3×3 convolved with a filter of size 2×2 and stride $(1, 1)$ will result in a feature map of size 2×2.

Note that sometimes, the filter does not perfectly fit the input image. If we decide to drop the part of the image where the filter doesn't fit and keep only the valid part of the image, we are using "valid padding". This results in a feature map that is reduced in size as compared to the input. On the other hand, if we want our feature map to keep the same size as the input, we use "same padding" which consists in "padding" the image with zeros before applying the filter, so that it fits correctly. Padding is another parameter of the convolutional layer.

Pooling layers are layers that aim at reducing the spatial size of a feature map while retaining important information that is useful to our task. This dimensionality reduction allows to reduce the number of parameters and therefore the computational power required to process the data. Furthermore, it is useful for extracting features which are rotational and positional invariant, meaning that small local movements in the input image don't affect the pooled outputs. In a convolutional neural network, the pooling layer is often added after the convolutional layer, more precisely after the non linearity application (for example RELU). The pooling layer operates on each feature map output by the convolutional layer, which results in a new set of the same number of pooled feature maps.

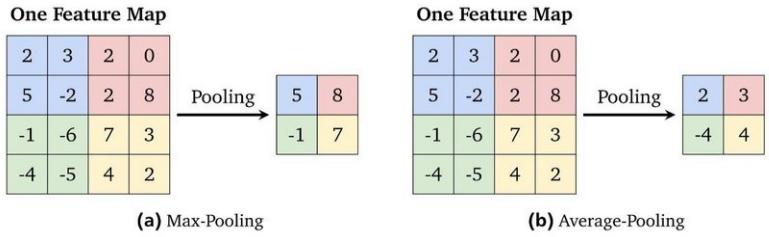


Figure 7: Average and Max Pooling illustration

Pooling works like a filter that is applied to feature maps. It is common to choose a filter of size 2×2 with a stride of (2, 2), which allows to halve each spatial dimension of the feature map. For example, a pooling layer applied to a feature map of size 6×6 will output a pooled feature map of size 3×3. To apply this "pooling" filter to the feature map, we need to choose a pooling operation. There exist two common types of pooling operations : Max Pooling and Average Pooling. Max Pooling returns the maximum value from each region in the image covered by the filter whereas Average Pooling returns the average value of each region.

2.1.2 Feature visualization

Convolutional neural network is a very powerful tool that allows to extract important and useful features from images in order to solve a specific task. However, it is not always clear how the network sees and interprets its input data, and recognizes special patterns or objects. Due to their lack of inscrutability, most deep learning models are often been referred to as "black boxes". For the past years, a lot of work has been put into making neural networks more interpretable to humans leading to major threads of research such as feature attribution and feature visualization.

To study how a CNN works in more detail, we can try to observe and understand what kind of features it learns throughout the network. Specifically, this is equivalent to observing the kind of images or patterns that significantly activate different parts of the network, for example a neuron at an individual position or an entire filter or even a layer as a whole. In order to do this, we can think about looking for examples in the dataset that induce a high activation of a particular neuron or filter. Another method called feature visualization by optimization was introduced in 2009 by Erhan et al. [5]. It consists in generating an "image" or pattern that specifically activates the part of the network we are interested in by optimizing its pixel values, starting from a random input image. In this section, we are going to detail feature visualization and more precisely focus on observing and interpreting what each individual filter is learning in a Convolutional Neural Network.

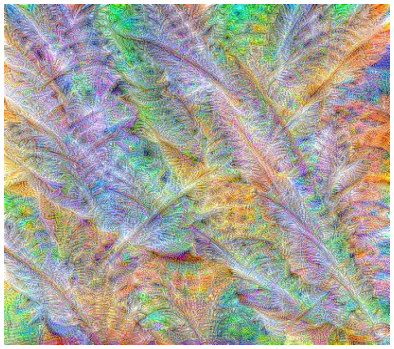
The goal of feature visualization by optimization is to find out what kind of input would cause a certain behavior (here, the activation of a particular filter) to better understand what the filter learns to detect. Usually, as described in the previous section, when you train a CNN, your input images and labels remain fixed and the network learns the weights of the filters using Backpropagation so that it predicts the right label. Here, we shift perspectives : we are no longer training a model to predict an image label. Instead, we are trying to generate an input image by learning its pixel values so that it maximizes the activation of a particular filter. In order to do that, we use derivatives to iteratively tweak the input towards maximizing the filter's output. This is made possible because neural networks are generally differentiable with respect to their inputs.

More precisely, the idea is the following : we start from a random input image, we choose a convolutional filter of interest in our pre-trained network and we apply Gradient Descent on the input image pixel values in order to maximize that filter's output, while leaving the Network's weights constant. If the network has been properly trained, filters activation should carry semantic meaning and induce interesting results. This will help us observe what kind of images turn a particular filter on and therefore to better understand what patterns it learns to detect. Note that since a filter's output (feature map) is technically a matrix, the actual function we are maximizing is the average of the feature map values.

Fabio M.Graetz worked on feature visualization and shared his observations in a blog publication. In his study, he performs feature visualization on several filters in different layers of a pre-trained VGG19 network. VGG19 architecture has in total 47 layers but only 19 layers with learnable weights

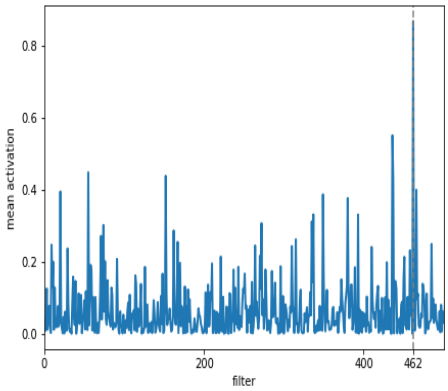
: 16 convolutional layers, and 3 fully connected layers. Here, he uses a pretrained version of the network trained for image classification on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories. As a result, it has learned rich feature representations for a wide range of images and therefore represents a good baseline for feature visualization.

As we can easily observe in figure 8, the generated patterns tend to be more complex as we go deeper into the network. The first layers generate simple visuals such as lines and curves. The closer you get to the output of the model, the more complex the textures and patterns become. The last layers generate more sophisticated and specific concepts in which we can even recognize parts of real life objects. For example, looking at the image generated by filter 123 of the layer 40, we can observe what seems to be a mix of animal eyes and maybe snouts. Looking at filter 462 of the layer 40, we can recognize a pattern that seems to correspond to a bird feather. Fabio M. Graetz shows that we can "test" these hypothesis (about what type of objects a specific filter responds to) by feeding a chosen image to the network and comparing its filters activation. We can see in figure 9 that feeding a bird feathers image to the model induces a very high mean activation of the filter 462 of the layer 40, as we expected. This confirms our visual recognition of a feather in the pattern generated in figure 9a.



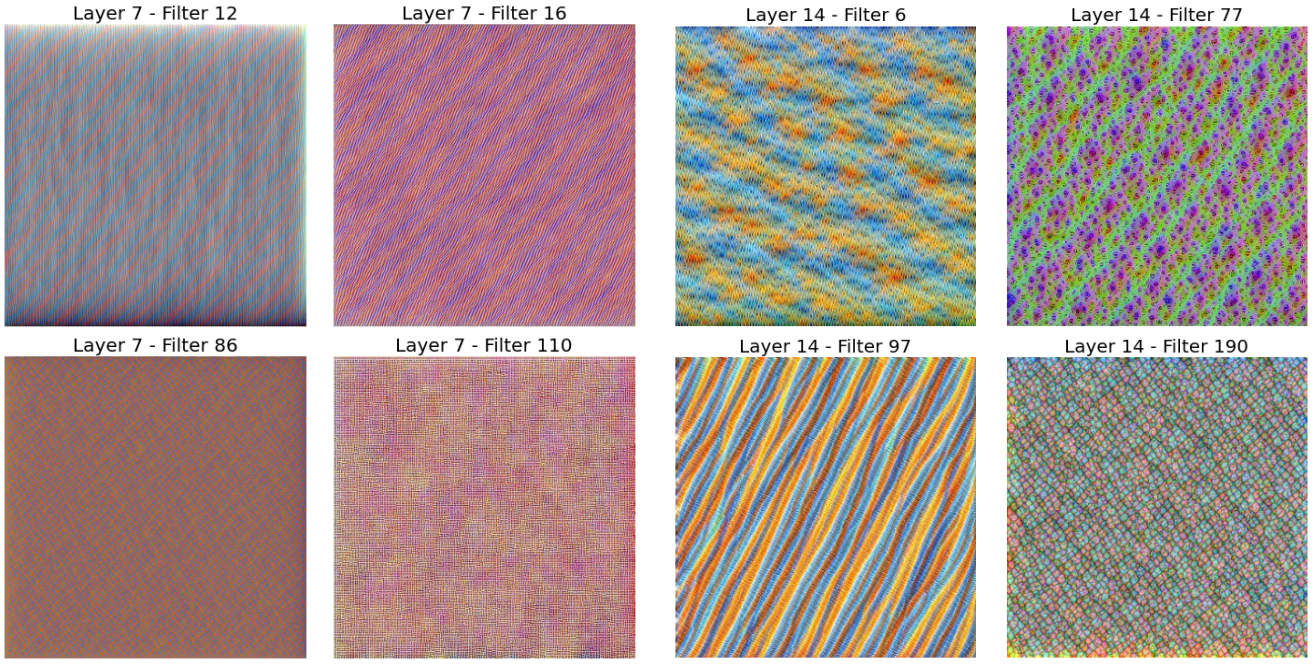
(a) Layer 40 - filter 462

(b) Bird feathers



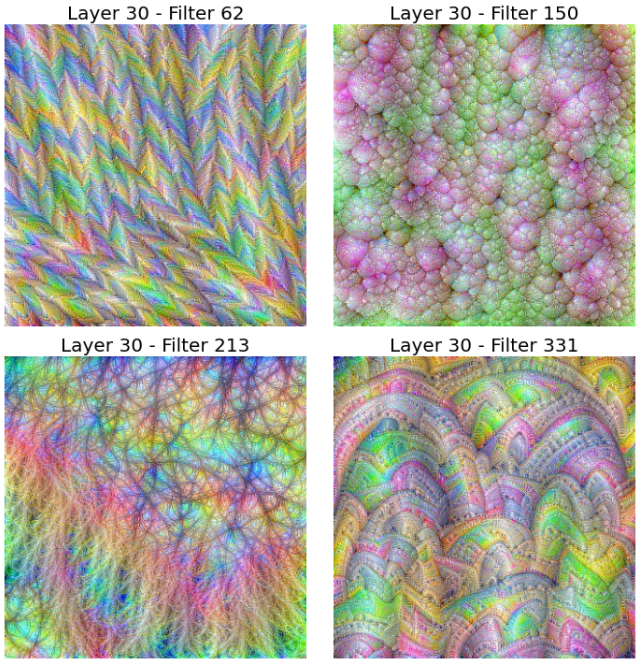
(c) Filters mean activation in layer 40 when feeding the bird feathers image to the model : high peak for the filter 462.

Figure 9: Recognizing a pattern

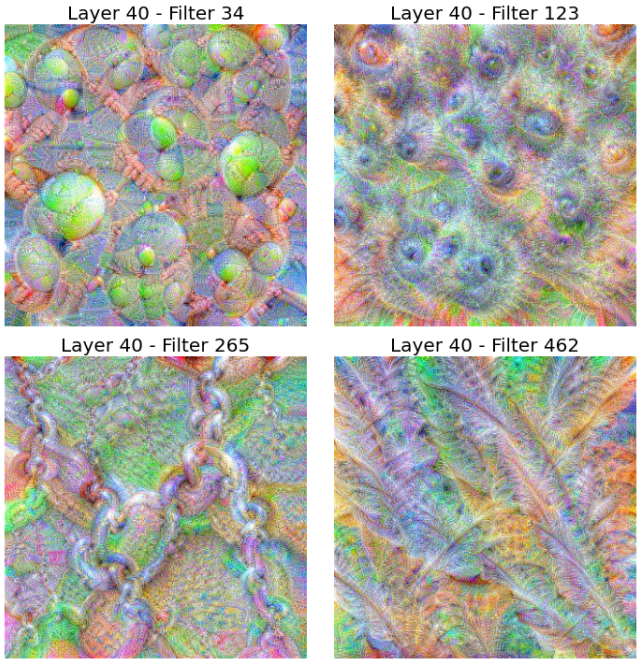


(a) Layer 7

(b) Layer 14



(c) Layer 30



(d) Layer 40

Figure 8: Feature visualization : several filters in different layers

2.1.3 State of the art CNN models

After having introduced the Convolutional neural networks, we will dicuss some of their useful applications. In this section, we define the three main tasks within Computer vision : image classification, object detection and image segmentation, and give some examples of State of the Art Convolutional neural networks that were developed to solve these tasks.

Image classification

ImageNet is a large dataset that contains over 15 millions labeled high-resolution images with around 22,000 categories, developed for computer vision research. The photographs were annotated by humans using crowdsourcing platforms. ILSVRC (ImageNet Large Scale Visual Recognition Competition) is an annual organized challenge that uses subsets from the ImageNet dataset, designed to encourage the development and benchmarking of state-of-the-art algorithms. More precisely, ILSVRC uses a subset of ImageNet of around 1000 images in each of 1000 categories. In all, there are roughly 1.3 million training images, 50,000 validation images and 100,000 testing images. In this section, our focus is on the challenge Image classification task that consists in predicting the classes of objects present in an image. We are going to review the models that have arisen from this challenge throughout the years, analyzing their differences and comparing their performance.

1. Alexnet

   One of the earliest convolutional neural networks developed was LeNet by Lecun et al. in 1998 [6] in order to classify digits in the famous dataset MNIST.
   In 2012, AlexNet developed by Alex Krizhevsky et al. [7] significantly outperformed LeNet and all the prior competitors, and won the ILSVRC 2012 challenge by reducing the top-5 error from 26% to 16.4%. AlexNet has a similar architecture as LeNet but deeper, with more filters per layer, and with stacked convolutional layers. It contains 11x11, 5x5,3x3, convolutions, max pooling, dropout and ReLU activations, and the last three layers are fully connected layers. The high performance of the model was justified by its depth, which was computationally expensive, but made possible due to the utilization of GPUS during training.

2. Inception (GoogleNet)

   The winner of the ILSVRC 2014 competition was Inception V1 (a.k.a. GoogleNet) by Szegedy et al. [8] with a top-5 error rate of 6.67%.
   The Inception architecture consists of a 22 layer deep CNN inspired by LeNet but contains a novel element called "inception module" from which the model gets its name. The inception module applies different types and sizes of convolutions to the same input, which allows to extract different kinds of features, and then stacks all the feature maps output together.
   However, its greatest innovation lies in its "factorised" convolutions that use 1×1 convolutions in order to reduce the number of parameters. This allows to drastically reduce the model size along with the risk of overfitting. In Inception, the total number of parameters is 4 million compared to 60 million in AlexNet.
   Moreover, nearly at the end of the network, Inception uses a global average pooling to average each feature map. The model also uses auxiliary classifiers for training : their losses are added to the total loss with a 0.3 weight. This allows to provide regularization and to avoid gradient vanishing problem.

3. VGGNet

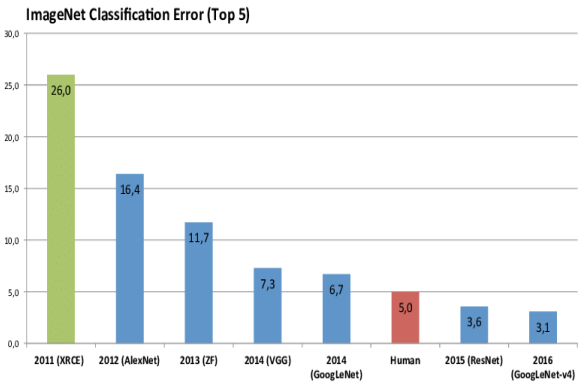
   The second winner of the ILSVRC 2014 competition was VGGNet (VGG16) developed by Simonyan et al. [9], with a top-5 error rate of 7.3 %.
   VGGNet was specially developed to reduce the number of parameters in convolutional layers and therefore to improve performance while reducing training time. There are multiple variants of VGGNet (VGG16, VGG19, etc.) which differ only in the total number of layers in the network. VGG16 consists of 16 convolutional layers and has a total of 138 million parameters.
   Unlike AlexNet and Inception, VGGNet only uses convolutional kernels of size 3x3 and more filters per layer. The idea behind having fixed size kernels is that 11x11, 5x5 and 3x3 convolutional kernels can be replicated by making use of multiple 3x3 kernels as bulding blocks. The replication is in terms of the receptive field covered by the kernels : we use small respective fields to replace a large one. This allows to reduce the number of parameters while keeping performance. For example the effect of one 5x5 conv layer (25 weights) can be achieved by implementing two 3x3 convolutional layers with a stride of one (18 weights).
   VGGNet is currently the most preferred choice in the community for extracting features from images. However, 138 million parameters can be a bit challenging to handle.

4. ResNet

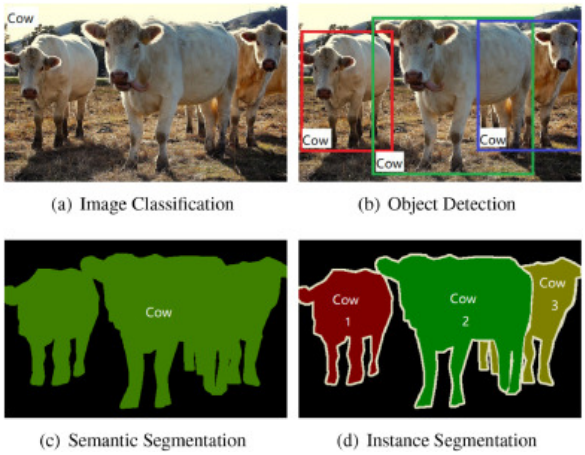
   At last, at the ILSVRC 2015, the Residual Neural Network (ResNet-152) by Kaiming He et al. [10] obtained a top-5 error rate of 3.57 %. In computer vision, the common idea is that in order to improve a network performance, "we have to go deeper". However, Kaiming He et al. [10] have observed that increasing network depth does not mean simply stacking layers together. Deep networks are hard to train because of the vanishing gradient problem : as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient very small. Therefore, as the network goes deeper, its accuracy gets saturated or even starts to degrade rapidly.
   In order to tackle this problem, ResNet adds "identity connections" between its layers : the network has to learn residual functions with reference to the layer inputs, instead of learning unreferenced functions, which makes the training easier.
   Residual blocks are stacked on top of each other to form the ResNet-152 network giving a total of 152 layers, however still having lower complexity than VGGNet (60 million parameters). Intuitively, these types of network are easier to optimize and can improve performance due to their increased depth.

5. DenseNet

   Huang et Al. [11] introduced Dense Convolutional Networks (DenseNets) in 2018 as convolutional networks that not only are deeper but also more efficient to train. DenseNet achieves comparable performance to ResNet while requiring significantly fewer parameters. For example, on ILSVRC, DenseNet-201 (20M parameters) obtains similar validation error as a 101-layer ResNet (40M parameters).
   While traditional CNN connect each layer to its subsequent layer, DenseNet, on the other hand, connects each layer to every other layer in a feed-forward way. This means that for each layer, the feature-maps of all previous layers are used as inputs and its own feature-maps are used as inputs into all subsequent layers.
   DenseNets have various advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature re-use, and drastically reduce the number of parameters. This allows DenseNets to obtain high performance on image classification benchmark tasks, while requiring less memory and computation than most state-of-the-art models.



(a) ImageNet Classification Error



(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

(b) Different types of tasks

Figure 10: State of the Art in Computer Vision

Object detection

Object localization refers to identifying the location of one or more objects in an image and drawing a bounding box around their extent. Object detection combines object localization with image classification : it localizes and classifies one or more objects in an image (see figure 10b).

Ross Girshick et al. [12] developed R-CNN in 2014, and later Fast-R-CNN [13], that both rely on selecting in "pre-process" a fixed number of region proposals thanks to a selective search algorithm, then extracting region features thanks to Convolutional neural networks and finally classifying each of these regions. In addition to predicting the presence of an object within the region, the algorithm also predicts four values (center coordinates, width, height) to help adjusting the bounding box of each region proposals. YOLO model was developed by Joseph Redmon et al. in 2015 [14] to solve Object detection. Unlike R-CNN and Fast R-CNN, YOLO is not a region-based algorithm : a single convolutional network predicts both the bounding boxes and the class probabilities for these boxes.

Image segmentation

Another important subject within computer vision is image segmentation. Segmentation identifies parts of the image and understands what object they belong to. Specifically, it divides an image into segments to simplify its analysis. Segments are set of pixels that represent objects or parts of objects. Another way to view segmentation is clustering, where pixels sharing features such as color, intensity, or texture are grouped together and represented as a single entity. Segmentation lays the basis for performing object detection and classification. There are two types of segmentation : semantic segmentation and instance segmentation.

Semantic segmentation classifies each pixels of an image into meaningful classes of objects. These classes are "semantically interpretable" and correspond to real-world categories. For example, you could isolate all the pixels associated with a car and color them in green (see figure 10b). Instance segmentation is a more complex task : it identifies each instance of each object in an image. It differs from semantic segmentation in that it doesn't categorize every pixel. If there are three cars in an image, semantic segmentation classifies all the cars as one instance, while instance segmentation identifies each individual car (see figure 10b).

One good approach for segmentation is the U-Net model developed in 2015 by Ronneberg et al. [15]. It was originally invented and used for biomedical image segmentation, but it can also be applied to other image-to-image translation tasks whose goal is to learn the mapping between an input image and an output image. The models consists in :

- a contraction path (also called encoder) that consists in a downsampling process through regular convolutions and max pooling layers. The spatial dimension of the image gradually reduces while the depth dimension (number of channels) increases. This part enables to extract important and useful features from the input image.
- an expansion path (also called decoder) that consists in a symmetric upsampling process through transposed convolutions along with regular convolutions. The spatial dimension of the image gradually increases while the depth dimension decreases. This part allows to recover the structural and localization information of the input image lost during the encoder part. For many image translation problems, there is a great part of low-level information shared between the input and output, they differ in surface appearance but usually share the same underlying structure. Therefore, at every step of the decoder, skip connections are used by concatenating the output of the transposed convolution layers with the feature maps from the encoder at the same level. The idea behind this is to use the features that have been learned while encoding the input image to decode it as an output image. It allows to preserve the structural integrity of the input image which reduces distortion and allows more precise locations.

The global architecture has a symmetric "U" shape (figure 11), which is where "U-Net" gets its name from.

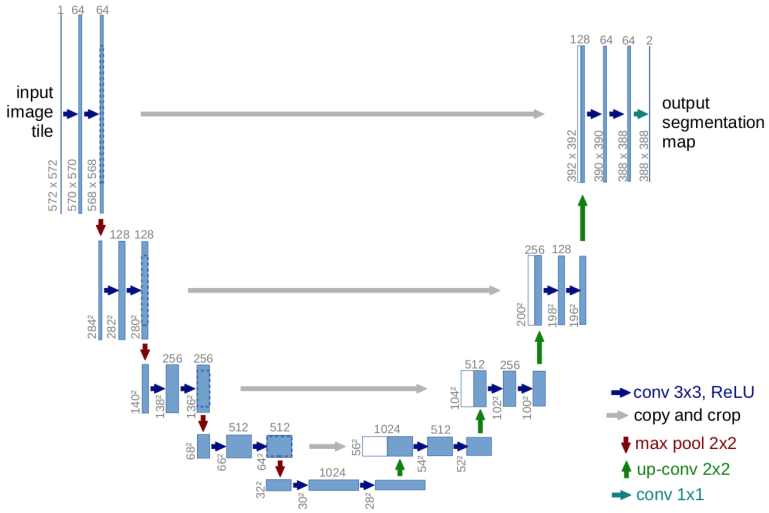
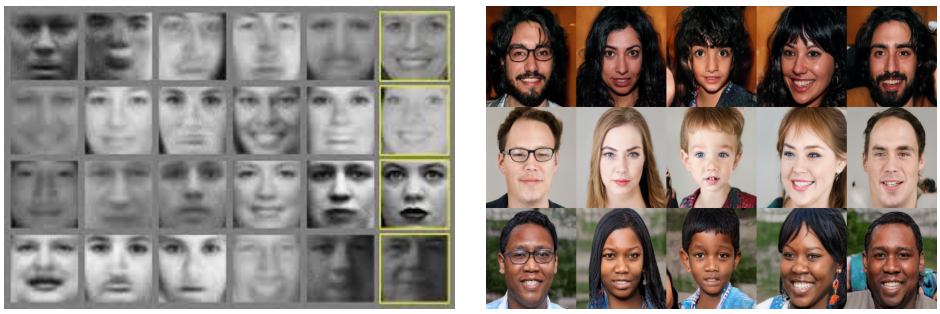


Figure 11: U-Net architecture

U-Net decoder is able to generate an image from a feature dense vector and can therefore be seen as a generative model. In the following section, we introduce the Generative adversarial networks (GANs), which also belong to the class of generative models.

2.2 Generative adversarial networks (GANs)

Described by Yann Lecun as "the most interesting idea in the last 10 years in Machine Learning", Generative adversarial networks (GANs) (GoodFellow et al. [16]) is a generative model that aims at sampling new data from a given data distribution learnt implicitly from a training set. What has specifically generated its interest and success during past years is its adversarial training.



(a) 2014 GANs (Goodfellow et al. [16]) (b) 2018 StyleGan (Karras et al. [17])

Figure 12: Images generation by GANs

2.2.1 Adversarial modeling framework

The main idea is that two networks, called the generator and the discriminator, are competing against each other : they have opposite goals during training. The generator aims at generating a new sample matching the data distribution (ie a "fake" sample) whereas the discriminator aims at detecting whether the sample in question is "real" or not (ie if it is coming from the training dataset or if it has been generated). This contest between the two entities allows to improve the quality of the data sampled from the generator network : the goal is to make sure that they match so closely the real data distribution that the discriminator is no longer able to make the difference. In other words, the generator is trained to get better at "fooling" the discriminator while the discriminator is trained to get better at discerning a real sample from a fake one. In particular, GANs have proved their effectiveness in Image generation, allowing to generate very high resolution images.

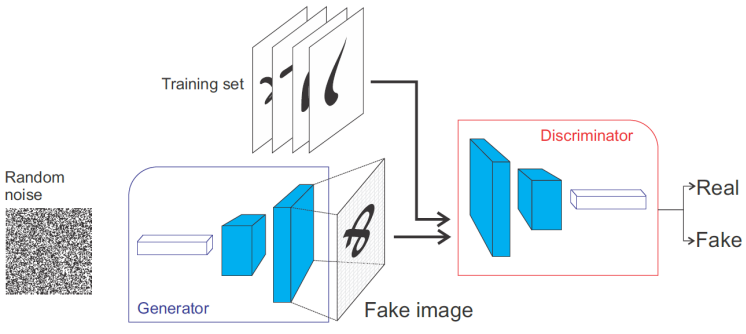


Figure 13: GAN architecture

This adversarial framework corresponds to a minimax two-player game between the generator :

$$G : Z \mapsto X$$

where

- $z \in Z$ represents an input vector of noise (uniform or Gaussian) associated to a prior $p_z(z)$
- $x \in X$ is a generated data point

and a discriminator :

$$D : X \to \{0, 1\}$$

where D(x) represents the probability that a data point x comes from the real data rather than from the generator. Therefore we train G to minimize the quantity $log(1 - D(G(z)))$ and we train D to maximize the probability of associating the right label to both training examples and samples from G. This is equivalent to solving the following minimax equation :

$$\min_G \max_D \; V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{1}$$

In game theory, the GAN converges when the discriminator and the generator reach a Nash equilibrium. A Nash equilibirum (NE) is a point (θ_D, θ_G) such that the cost function $V(D, G)$ is at a minimum with respect to θ_D and $V(D, G)$ is at a minimum with respect to θ_G.

In their publication, Goodfellow et al. (2014) [16] prove that this minimax game has a unique solution when (G^*, D^*) satisfy :

$$\begin{cases} p_{G^*} = p_{data} \\ D^*(x) = \dfrac{p_{data}(x)}{p_{data}(x) + p_G(x)} \end{cases} \iff \begin{cases} p_{G^*} = p_{data} \\ D^*(x) = \dfrac{1}{2} \end{cases}$$

where p_G is the distribution of the samples $G(z)$ obtained when $z \sim p_z$ and p_{data} is the data distribution.

In practice, however, GANs have been shown to struggle to reach this NE, we will discuss this in the following section 2.2.2.

If we consider the case where the generator G and the discriminator D are neural networks (called adversarial nets), the GAN can be trained with backpropagation with alternating gradient descent between G and D. The algorithm is the following :

Algorithme 1 : GAN training

for *epochs* $1, \ldots, N$ **do**

- Sample minibatch of size m from data : $x^{(1)}, \ldots, x^{(m)} \sim p_{data}$
- Sample minibatch of size m of noise : $z^{(1)}, \ldots, z^{(m)} \sim p_z$
- Take a gradient ascent step on the **discriminator** parameters :

$$\nabla_{\theta_d} V(G_{\theta_g}, D_{\theta_d}) = \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [log(D(x^{(i)}) + log(1 - D(G(z^{(i)})))]$$

- Take a gradient descent step on the **generator** parameters :

$$\nabla_{\theta_g} V(G_{\theta_g}, D_{\theta_d}) = \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(z^{(i)})))$$

end

2.2.2 Limitations

Despite their wide success, GANs can be very difficult to work with as they face many challenges and limitations, especially due to their unstable training.
In the past few years, a significant amount of work has been put into trying to solve these issues : new architectures, regularization methods, noise perturbations, modified losses...
We discuss here the main sources of instability and some of the methods that have been proposed to stabilize the training of GANs.

Mode collapse

Mode collapse is one of the complex problems that a GAN can encounter. Real-world data distributions are highly complex and multimodal, therefore the probability distribution describing the data has multiple “peaks” where different sub-groups of samples are concentrated. Naturally, you want your generator to capture this diversity in the data distribution and to be able to produce this variety of modes.

The mode collapse in GANs refers to the problem of missing some of the modes of the multi-modal data it was trained on. This happens when the generator believes that it can fool the discriminator by locking on to a single mode. That is, the generator produces similar samples, exclusively from this mode. More intuitively, imagine that your generator finds a plausible output that succeeds in fooling the discriminator, it may ”be happy” and therefore learn to produce only that one output. If the generator starts producing the same type of outputs over and over again, the discriminator eventually figures out that samples from this mode are fake and rejects them. As a result, the generator simply locks on to another mode. This goes on indefinitely, and leads to limited diversity among the generated samples.

Failure to converge

Finding Nash equilibria is known to be difficult, in particular in the GAN case where the cost functions are non-convex and the parameters very high-dimensional. Some previous approaches to GAN train-

ing consider that a Nash equilibrium occurs when each player has minimal cost and therefore apply traditional gradient descent to minimize each player's cost simultaneously. However, Salimans et Al. [18] explain that we have no guarantee that this procedure will be able to converge. The networks parameters are updated separately but a modification to the parameters θ_D that aims to reduce $V(D)$ could induce $V(G)$ to increase, and vice versa. Therefore, gradient descent fails to converge for many games. In a more intuitive way: one opponent always countermeasures the other opponent actions which makes the models harder to converge.

We have seen earlier that the networks are opposed : if the generator improves while training, the discriminator performance gets worse because it becomes harder to distinguish fake samples from real data. At the global optimum, the generator achieves its best performance and the discriminator has a 50 % accuracy. This corresponds to randomness : the discriminator flips a coin to make its prediction, so its feedback gets meaningful. If we continue training the GAN beyond this point, the generator will be trained based on poor feedback, and it might end up degrading the quality of its outputs. Therefore, it is important to stop the training once the discriminator has 'lost' the game.

Vanishing gradients

If the discriminator is too good, then generator training can fail due to vanishing gradients. More precisely, if it becomes very accurate and predicts $D(x) = 1 \; \forall x \in X$ and $D(G(z)) = 0 \; \forall z \in Z$, the loss function V will approach 0, producing gradients close to zero as well. Therefore the generator will be trained on very little feedback and will have difficulty in updating its weights, which may cause the slowing or the stopping of the learning. This is known as vanishing gradients problem. To avoid this problem, Goodfellow et al. (2014) [16] proposed to use an alternative loss for the generator, namely $\log D(G(z))$, which is widely used in practice and referred to as non-saturating (NS) loss.

2.2.3 Conditional GANs

In their paper, Mirza et Al. (2014) [19] introduce a conditional version of GANs named conditional GANs (cGANs). Just as GANs learn a generative model of data, cGANs learn a conditional generative model. Specifically, the generator and the discriminator are conditioned on some additional information $y \sim Y$ (for example class labels or data from other modalities). The idea is to feed the information y we wish to condition on, into both the discriminator and generator as additional input layer (by concatenation).

Conditioning the model on extra information allows to better control the data generation process, which can be very useful for example if you want to generate an image of a desired class on demand. On the MNIST dataset, they prove that they can have control over the digits they generate by conditioning on the class labels. For example, if we condition on the label "0", the generator will generate samples that correspond to the "0" digit.

More precisely, we consider a conditional generator :

$$G_c : Z \times Y \mapsto X$$

and a conditional discriminator :

$$D_c : X \times Y \mapsto \{0, 1\}$$

which are trained against the following objective :

$$\min_G \max_D \; V_{cGAN}(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x|y)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z|y)))] \tag{2}$$

In the generator G, the prior input noise $p_z(z)$ and y are combined in joint hidden representation, which forces the generator to generate images in adherence to the condition y. In the discriminator D, the ground truth image as well as the extra information y are presented as inputs to a discriminative function, which forces D to declare as "fake" any image generated by G that does not match the condition y.

Interestingly, Conditional GANs are totally suitable for image-to-image translation tasks where we are aim at "translating" an input image into a corresponding output image. Indeed, we can condition our cGAN on an input image and train it to generate the corresponding target image. In 2018, Isola et al. [20] introduced the Pix2pix model that applies Conditional GANs to image-to-image translation tasks. We will develop Pix2pix in more detail within section 3.2.

2.3 Transfer Learning

Transfer Learning refers to methods that allow transferring knowledge acquired while solving a specific problem to solve a different but related problem. Re-using the model weights from a pre-trained model on a different but related task was proved to improve drastically both performance and efficiency. Consider this well-known intuitive example : you want to learn to play the piano, and you already know how to play the guitar. You will use your music skills to learn this new instrument and it is very likely that it will take you less time than someone starting music from scratch.

We will try to give a more formal and general definition. Here, we consider that a machine learning problem is defined by :

- A domain $\mathcal{D} = \{\mathcal{X},\ P(X)\}$
  where $\mathcal{X}$ is a feature space and $P(X)$ with $X = \{x_1, \ldots, x_n\} \in \mathcal{X}$ is the probability distribution associated to $\mathcal{X}$
- A task $\mathcal{T} = \{\mathcal{Y},\ f(.)\}$
  where $\mathcal{Y}$ is a label space and $f(.)$ is an objective prediction function learned from the training data $\{x_i,\ y_i\}$ with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$

Now let's consider two machine learning problems :

- the source S associated to the domain $\mathcal{D}_S$ and the task $\mathcal{T}_S$
- the target T associated to the domain $\mathcal{D}_T$ and the task $\mathcal{T}_T$

with $\mathcal{D}_S \neq \mathcal{D}_T$ and $\mathcal{T}_S \neq \mathcal{T}_T$.

The goal of transfer learning is to improve the learning of the predictive objective function of our target model T based on the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$.

In the specific case of deep learning, transfer learning is a technique where a neural network model is first trained on a problem similar to the problem we want to solve. One or more layers from the trained model and their associated weights are then used in a new model trained on the problem of interest. Transfer learning is very flexible : there are many creative and useful ways of using pre-trained models. We will discuss some of them.

2.3.1 Feature Extractor

Let's recall that Convolutional neural networks allow to learn compressed feature representations of images, capturing their properties. As we have seen in section 2.1.2, in a CNN basic architecture, feature visualization has shown that first layers learn low-level features such as rough textures, edges, abstract patterns and as we go deeper, the network tends to learn more complex features. Layer close to the output produce features specific to the task. This gives an idea of why transfer learning can be interesting when working with images and CNN models. We can use pre-trained CNN models to extract their learned image features and re-use them in a similar context. We can choose which layers (which features maps) of the pre-trained model would be useful for training our new model. It depends on the task and on the pre-trained model we use.

For example, let's consider a pre-trained model trained on ImageNet to classify objects in images. If the task is quite different, we may want to cut out the last layers and use the features maps generated by the middle layers of the pre-trained model. On the other hand, if the task is quite similar (image classification), then perhaps we can use the features extracted from much deeper layers of the pre-trained model. It goes without saying that a pre-trained model can also be used as-is, without making any change and without further training, if the tasks are the same. This is what we call feature extraction models : we use pre-trained models to extract interesting features. Here, we differentiate two types of features extractions :

- Standalone feature extraction : The pre-trained model is used as a separate feature extraction program. Here, the idea is to use transfer learning as a technique of pre-processing : for example, we represent each of our input images by a features vector based on the pre-trained model selected features maps. We then give these features vectors as input to our new model and start the training.
- Integrated feature extractor : The pre-trained model, or a desired portion of the model, is directly integrated into a new neural network model. The weights of the pre-trained chosen layers can be frozen, meaning that they will not be updated as the new model is trained : their values remain fixed.

2.3.2 Weight initialization

We can also use transfer learning as a technique of weight initialization : the pre-trained model, or a desired portion of the model, is integrated into a new neural network model but this time the weights are updated during the training of the new model, often with a lower learning rate. This is a good option if the data used for the pre-trained model differs a lot from the data you train your new model on or if the task is very different. The weights in re-used layers are used as the starting point for the training process and adapted in response to the new problem.

2.3.3 Pre-trained models

Over the last decade, high-performing "state of the arts" models have been developed in computer vision for various tasks such as image classification, segmentation or object detection. In particular, ImageNet challenge for image classification has induced several innovations in the architecture and training of convolutional neural networks and allowed the emergence of models such as VGG, Resnet, GoogleNet... These models have provided rich image features that were trained on more than 1,000,000 images for 1,000 categories. However, most of these models require a high training time and important resources. Fortunately, many of them have been released publicly and are easily accessible : many

libraries provide APIs to download and use the models directly. In particular, this is very useful to allow transfer learning in computer vision applications using these models as basis.

2.3.4 Example : U-Net on steroids

In 2018, the fourth Kaggle Data Science Bowl was organized, its goal was to identify nuclei of cells in a given microscopy image and to provide masks for each nucleus independently. Along with data augmentation and important post-processing, the winning team mainly owes its impressive results to their approach "U-Net on Steroids". Specifically, they used a network architecture based on transfer learning ; a U-Net-like encoder-decoder architecture with encoder pre-trained on ImageNet. The idea is that having been trained on a significant amount of data, the pre-trained encoder learns better feature representations of input images and therefore leads to improved results. Their choice of encoders included very deep models such as Resnet-152, InceptionResnetV2 and Resnet101. This has allowed them to boost up their model performance and to achieve a F1-score of 0.6312, beating the second-placed entry by impressive 0.017.

3 Fluorescence images prediction

In silico labeling consists in predicting the fluorescence labeling of cells. This task can be seen as image-to-image translation whose goal is to learn the mapping between an input image and an output image. You want to "translate" your input image into a corresponding output image. In our case, the input is the transmitted light image and the target is the fluorescence image. We are trying to learn a model that would be able to generate fluorescent images from transmitted light associated images. More specifically, it can be seen as a regression task where the goal is to map pixels to pixels, we want to predict the correct value of each pixel of our target image. Throughout this section, we present our approach to In Silico Labeling task for predicting our fluorescence images and we discuss the results we obtained.

3.1 U-Net approach

In their paper, Ounkomol et al. [3] consider a U-Net based model for their In silico labeling task. This is also the architecture we selected for our In Silico model. Specifically, we thought that U-Net on Steroids could be an interesting and suitable approach for this task. Therefore, we initialized the weights of our encoder with the weights of a Resnet101 model trained on ImageNet. As expected, this had improved our performances. However, Resnet101 being a very heavy model which contains more than 40M parameters, we searched for a more efficient model. The final model we consider for our In Silico Labeling task is a U-Net on steroids whose encoder is a DenseNet121 model pre-trained on ImageNet.

Our initial naive approach was to train our U-Net based model by using a L1 or L2 loss function: we asked the model to minimize the Euclidean distance between predicted and ground truth pixels.

$$\mathcal{L}_{L1} = \frac{1}{n}\sum_{i=1}^{n} |\widehat{y}_i - y_i| \tag{3}$$

$$\mathcal{L}_{L2} = \frac{1}{n}\sum_{i=1}^{n} (\widehat{y}_i - y_i)^2 \tag{4}$$

However, we observed that these losses tend to cause blurry images. This is because Euclidean distance is minimized by averaging all plausible outputs. In order to tackle this issue, Isola et al. (2018) [20] propose to learn a loss function adapted to the image-to-image translation task along with the map function between the input image and the corresponding output image.

3.2 Pix2pix loss

In their paper, Isola et al. [20] propose to apply adversarial training to image-to-image translation problems thanks to their pix2pix model based on Conditional GANs (CGAN) [19] (see section 2.2.3). The idea is to condition the GAN with the input image and to train it to generate the corresponding output image. More precisely, the generator receives an input image and learns to predict the corresponding target image. The generator optimizes the same GAN objective as usual except that here, its objective is not only to fool the discriminator but also to be as near as possible to the ground truth output in a pixel wise manner. Therefore, a L1 loss component is added to its objective loss. The discriminator's task, on the other hand, stays the same.

The final objective is the following :

$$\min_G \max_D \ \mathcal{L}_{pix2pix}(D,G) = \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G) \tag{5}$$

with :

- $\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{y\sim p_{data}(y)}[logD(y|x)] + \mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z|x))|x)]$
- $\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(z|x)\|_1]$

For the generator architecture, an encoder-decoder network is used, typically a U-net model, for the same reasons that we explained earlier. As for the discriminator, the authors use a fully-convolutional network that they called "PatchGAN". Usually, a GAN discriminator maps an input image to a single scalar output that indicates whether the image in question is "real" or "fake". A PatchGAN, on the other hand, maps an input image to a $N \times N$ grid of outputs :

$$grid_{D_{patch}} : X, Y \to [0,1]^{N\times N}$$
$$(x,y) \mapsto grid_{D_{patch}}(x,y)$$

If we trace back their receptive field, each grid element corresponds to a particular patch in the input image and allows to classify this patch as "real" or "fake". The motivation behind this PatchGAN discriminator is to penalize structure at the scale of patches in order to encourage modeling the high-frequencies in the image. We consider that low-frequency structure is captured by the L1 term in the generator loss. To provide the final output of the PatchGAN discriminator and to classify the whole image as real or fake, the $N \times N$ output elements are averaged.

$$D_{patch}(x,y) = \frac{1}{N\times N} \sum_{i=1}^{N} \sum_{j=1}^{N} [grid_{D_{patch}}(x,y)]_{i,j} \tag{6}$$

As intuitively explained by Isola himself, this can be seen as equivalent to manually cropping the input image in $N \times N$ patches, running a regular discriminator over each patch and averaging the results.

3.3 Perceptual loss : content and style representations

3.3.1 Style transfer

In their famous paper, Gatys et Al. [21] tackle texture transfer problem by presenting a new optimization-based method called "Neural Algorithm of Artistic Style" able to produce images that combine the content and the style of two different images. Texture transfer can be considered as transferring the style from one image onto another, also known as image style transfer.

To be able to manipulate content and style, you first need to find a way to extract independently the content and the style information within an image. Their principal idea is to use the feature representations learned by high performing state-of-the-art CNNs, in other words they use feature extraction by transfer learning (see section 2.2). In particular, they use VGG network trained on object recognition and localisation to extract its deep image representations.

Content

A given input image is encoded in each layer of the Convolutional Neural Network by the filter responses to that image, called feature maps. More formally, we can store features maps in a matrix $F^l \in \mathcal{R}^{N_l \times M_l}$ where :

- l is the layer l of the network
- N_l is the number of features maps in layer l (usually increases while going deeper)
- M_l is the size of each of the feature maps of layer l, it is given by the height times the width of the feature map (usually decreases while going deeper)
- $F^l_{i,j}$ is the activation of the i^{th} filter at position j in layer l

Gatys et Al. [21] first study consists in visualizing these feature maps at different processing stages in the CNN. In order to do that, they perform a gradient descent on a white noise image to find another image that matches the feature responses of the original image.

More precisely, let's denote c the input image and x the reconstructed image, and C^l and F^l their feature representation in layer l. Starting from a white noise image, we perform gradient descent to progressively update the reconstructed image so that it minimizes the squared-error loss $\mathcal{L}_{content}$ between the two feature representations. We want the reconstructed image x to match the feature responses of the input image c.

$$\mathcal{L}_{content}(c, x, l) = \frac{1}{2} \sum_{i,j} (F^l_{i,j} - C^l_{i,j})^2 \tag{7}$$

They apply this to different layers of the VGG network and compare the reconstructed image they generate to the original image. They observe that lower layers tend to reconstruct the input image with exactitude, in a pixel-wise manner. Higher layers, on the other hand, allow to preserve image content and overall spatial structure but not color, texture or exact shape. They lose the detailed pixel-information but preserve the high-level content of the input image. Therefore, for the rest of their study, they use the feature representations given by one of VGG deep layers to extract the content information in an image.

Style

As for the style content, they introduce a new feature space based on statistical properties derived from the features, designed to capture texture information of an image. The idea is to compute correlations between the different features of different layers in the CNN. These correlations are computed thanks to the Gram matrix $G^l \in \mathcal{R}^{N_l \times N_l}$ defined by :

$$\begin{aligned} G^l_{i,j} &= < F^l_{i,;} \, , F^l_{j,:} > \\ &= \sum_k F^l_{i,k} F^l_{j,k} \end{aligned} \tag{8}$$

To visualize these style feature representations, they perform a gradient descent on a white noise image to construct another image that matches the style representation of an input image. Let's denote s an input image that has a particular style and x the generated image we reconstruct, and S^l and G^l their respective Gram matrices. We modify x until it minimizes the mean-squared error l_{style} between the two style representations :

$$l_{style}(s, x, l) = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{i,j} - S^l_{i,j})^2 \tag{9}$$

If we consider multiple layers then the total style loss to minimize is :

$$\mathcal{L}_{style}(s, x) = \sum_{l=0}^{L} w_l \; l_{style}(s, x, l) \tag{10}$$

where w_l are weights that indicate the contribution of each layer to the total style loss.

They observe that performing style reconstruction at several layers at once enables to capture the image texture at different scales without capturing its spatial structure. Therefore, for the rest of their study, they use the Gram matrices constructed from the features of multiple VGG layers as the "style representation" of an image.

Based on these observations, Gatys et al. [21] developed a method able to generate images that combine the content and style from two different images. The idea is similar to the one described before. They generate a white noise image x and update it by gradient descent in order to jointly minimise the content loss between x and a content image c, and the style loss between x and a style image s. The total loss is a weighted sum of the two losses :

$$\mathcal{L}_{total}(c, s, x) = \alpha \; \mathcal{L}_{content}(c, x) + \beta \; \mathcal{L}_{style}(s, x) \tag{11}$$

where α and β are the weights associated to content and style reconstruction, respectively. Intuitively, the weights allow to control the balance between content and style : a high value of α produces an image that matches the content with little influence of the style and a high value of β produces an image that matches the style but has difficulty to represent the content. This type of loss based on high-level features extracted from pre-trained networks is commonly named "perceptual loss".

(a) Content (b) Style

(c) Content + style

Figure 14: Style transfer illustration

3.3.2 Perceptual loss for image-to-image translation

Johnson et al. [22] propose to adapt the approach presented by Gatis et al. [21] to tackle image-to-image translation problems. The idea is to train a network that aims at minimizing a perceptual loss between the output and ground truth images, rather than a per-pixel loss. Besides producing blurry images, per-pixel losses depend only on low pixel information and therefore are week at measuring images similarities : they don't capture "perceptual" differences between images. For example, if we consider two identical images that differ only from one or few pixels, their per-pixel loss would be high even though they look very much alike. Perceptual losses capture image similarities more robustly.

In their paper, Johson et al. [22] propose to train a network for image-to-image translation task using a perceptual loss that depends on high level features from a pre-trained model like the one defined by Gatis et al. [21] In their experiments, they prove that this approach works well on image-to-image translation by trying to generate high-resolution images from low-resolution images. They also apply their method to style transfer and are able to generate the same results as Gatis et al. [21] but three orders of magnitude faster.

They define an image transformation network, typically a U-Net model, parameterized by its weights W, that learns a mapping function f_W between input and output images. The network is trained using stochastic gradient descent to minimize a weighted combination of loss functions :

$$W^* = \arg\max_W \mathbb{E}_{x,\{y\}} \left[\sum_i \lambda_i l_i(f_W(x), y) \right] \tag{12}$$

The losses l_i depend on your task. In our case, we are not interested in style transfer, we are interested in transforming an input image into the corresponding output image. More precisely, we want our model to learn to reconstruct the content of an image but not necessarily its style. For our task, we used a weighted combination of the following losses :

- The perceptual content loss $\mathcal{L}_{content}$ defined in the previous section that encourages our output image to capture perceptual similarities with the target image. This perceptual loss allows to increase the images quality, with less blurring.
- A pixel loss defined by the normalized Euclidean distance between the output and the target image, since we have a ground-truth that the network is expected to match.
- A total variation regularization loss defined by the sum of the absolute differences for neighboring pixel-values in an image. This measures how much noise is in an image and can therefore help us decreasing noise in the output image.

Naturally, if we were trying to realize a style transfer task such as combining the style and the content of two images, we would add the style loss, and drop the pixel loss. In our experiments, we weighted the content loss with a 0.8 weight, and both pixel loss and total variation loss with a 0.1 weight.

3.4 Datasets and pre-processing

We work on two different datasets. For our first dataset, transmitted light and fluorescence microscopy are performed at 16 non-overlapping fields of view (fov) per well, for a total of 60 wells. This produces an experimental dataset of 16x60 = 960 images each of size 2160x2160. Each fov can contain one or several cells, or sometimes no cell. Both fluorescence microscopy image (here, DAPI image) and the corresponding label-free transmitted light images (Brightfield and phase contrast) are acquired

for each fov. This actually gives a total of 960x3 = 2880 images in our dataset. Note that Brightfield images and fluorescence images are 3D images obtained by focal plane merging, also known as z-stacking : they combine multiple images taken at different focus distances to give a resulting image with a greater depth of field. In our case, they combine 4 "z-stacks".The fluorescence stain used here is DAPI. DAPI is a blue fluorescent dye that labels a cell's nuclear membrane and binds to regions of DNA rich in A-T base pairs. DAPI is effective in highlighting the cell nucleus, where the DNA is housed. Based on this dataset, the goal is to develop an In Silico Labeling model capable of predicting the fluorescent DAPI image from the transmitted light images (Brightfield and Phase contrast).

Our second dataset contains 100 images only. Here, the image resolution is smaller with images of size 1024x1024. Both fluorescence microscopy images and the corresponding label-free DIC image are acquired. Specifically, each image has 4 fluorescence counterparts thanks to 4 different fluorescent stains : Hoechst, Cy5, Cy3 and GFP. Each image is a 3D image that consists in 5 z-stacks. Like DAPI, Hoechst is another fluorescent dye used to stain DNA and to label a cell nucleus. Cyanine3 (Cy3) and cyanine 5 (Cy5) are widely used dyes belonging to a common family. Here, Cy5 provides "cell masks" that label the cells' membrane. Cy3, on the other hand, is here conjugated to a protein that is expressed specifically during the G1 stage of the cell cycle, which allows the fluorescence labeling of cells that are in G1 stage. Finally, here, GFP is a green fluorescent protein that we use to mark a specific protein expressed during S/G2 phase in order to label cells that are in the S/G2 phase of the cell cycle. These two last fluorescent stains both allow to label stages of the cell cycle. We use this dataset especially for trying to predict Cy3 and GFP fluorescence channels from DIC images.

Our last task consists in nuclei segmentation. In order to do this, we use the first dataset and create ground truth segmentation masks by applying Cellpose pre-trained model to our DAPI images. After visually checking the quality of the generated masks, we only select images that have accurate nuclei masks. This results in a final dataset of 924 images in total. The goal here is to develop a segmentation model able to predict nuclei masks from transmitted light images (Brightfield and Phase contrast).

For each of our task, the dataset is split in train, test and validation sets, and the same preprocessing is applied. Each image is normalized with a z-score normalization. Specifically, we apply the following normalization to each pixel of the image :

$$new\_pixel = \frac{pixel - \mu}{\sigma} \tag{13}$$

where μ is the mean of the image and σ its standard deviation.

Due to memory problems, we train all of our models on 512x512 crops randomly taken within each image instead of training models directly on the images. We also use some data augmentation by randomly applying flips and rotations to images. This allows to increase the size of our dataset and its diversity, and therefore to reduce overfitting.

3.5 Results

To tackle our In silico labeling task, we choose to apply a U-Net architecture. We compare the use of a classical U-Net with a pre-trained U-Net (U-Net on steroids). We train our models with the different losses presented in the previous section and compare their performance. We present the results obtained on our two datasets in this section, as well as the challenges encountered.

To evaluate our models, we use the Pearson correlation coefficient (PCC) which is a metric that allows to measure the linear correlation between our predicted image y and its ground truth x. More

specifically, PCC is the covariance of the two variables divided by the product of their standard deviations.

$$r(x,y) = \frac{\sum_{i=0}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=0}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=0}^{n}(y_i - \bar{y})^2}} \tag{14}$$

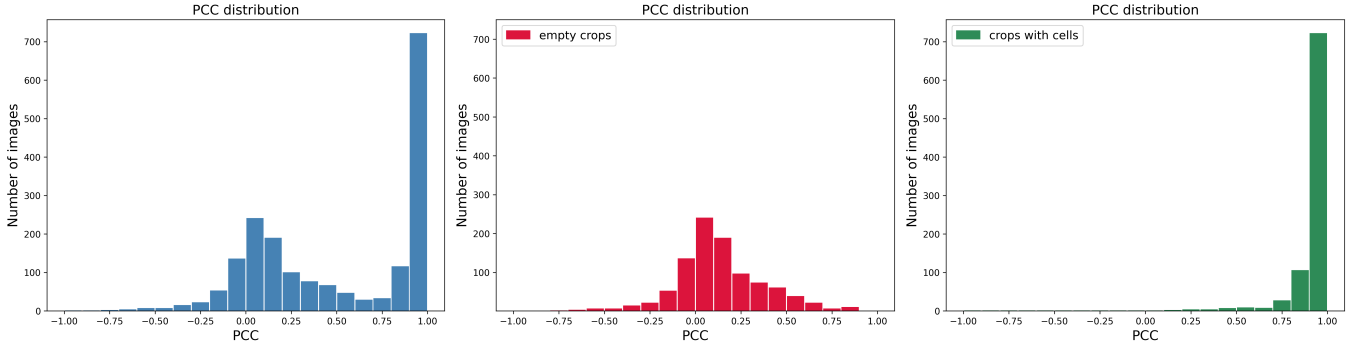
Pearson correlation coefficient allows to measure the similarities between two images. The higher the PCC is between our predicted images and the ground truth images, the better our model performs.

3.5.1 Fluorescence prediction from Brightfield and Phase contrast images

In this section, our goal is to develop a model able to predict DAPI fluorescence images that label the nucleus within each cell, from our transmitted light images.

Our first approach consisted in a U-Net model trained on 5540 crops and tested on 1850 crops, with a simple L1 loss. When observing results, our model seemed to perform already pretty well with predictions visually close to the ground truth fluorescence images. However, at our surprise, the mean pearson coefficient correlation evaluated on the train set was only of 0.52681 and of 0.51215 on the test set. We studied our results in more detail and observed that the PCC had a very high standard deviation (0.43560 on the test set), meaning that the model performed very well on certain images of the test set and very poorly on others. Specifically, after calculating PCC for each predicted image, we noticed that the model had difficulty in predicting empty crops, ie that contain no cell. The model tends to predict a great amount of noise instead of predicting an empty image that matches the ground truth, resulting in a very low PCC. This problem is counter-intuitive : while our model performs well on the complex task of generating fluorescence images of cells, it can be hard to imagine that it has trouble to learn such a simple relation as predicting an empty image whenever it detects no cell in the transmitted light image.

Our test set is approximately divided into 970 empty crops and 880 crops containing at least one cell. Figure 15 shows the PCC distribution on the test set and confirms our visual observations. The model performs well on images that contain cells, with a mean PCC of 0.926 and a standard deviation of 0.098. On the other hand, it has real difficulty to learn empty crops with a mean PCC of 0.136 only and a standard deviation of 0.235. Figure 16 shows examples of empty and non-empty crops.



(a) PCC on the whole test set (b) PCC on empty test crops (c) PCC on non-empty test crops

Figure 15: PCC distribution on the test set

To tackle this problem, we tried different solutions. We went back to questioning our pre-processing and tried to change the image normalization. We tried a z-score normalization common to each image, with statistics calculated on the whole train set. We also tried a min-max normalizaton which consists in re-scaling each pixel value between 0 and 1. Furthermore, we tried weighting our model loss in function of the number of cells a crop contains. Finally, we also questioned our metric choice.

We thought about using a different image similarity metric than the pearson correlation coefficient. One good way of checking the quality of generated images is to apply both predicted and ground truth images to a benchmark task and compare the performances of a pre-trained model. If it predicts similar outputs in both cases, this means that our model is able to generate good images that are close to ground truth. Here, we used Cellpose model (Stringer et al. [23]) on a nuclei segmentation task. We compared the nuclei masks obtained from our generated DAPI images, with the nuclei masks obtained from ground truth DAPI images. Specifically, we evaluated the Jaccard index between the two segmentation masks (see Jaccard definition in section 4.1) . If it is close to 1, the two masks are similar meaning that the two DAPI images are likely to be similar as well. However, the results obtained with this metric were globally consistent with those obtained with PCC, except for some interesting examples that gave a very good Jaccard Index despite a very low PCC (see Figure 16).

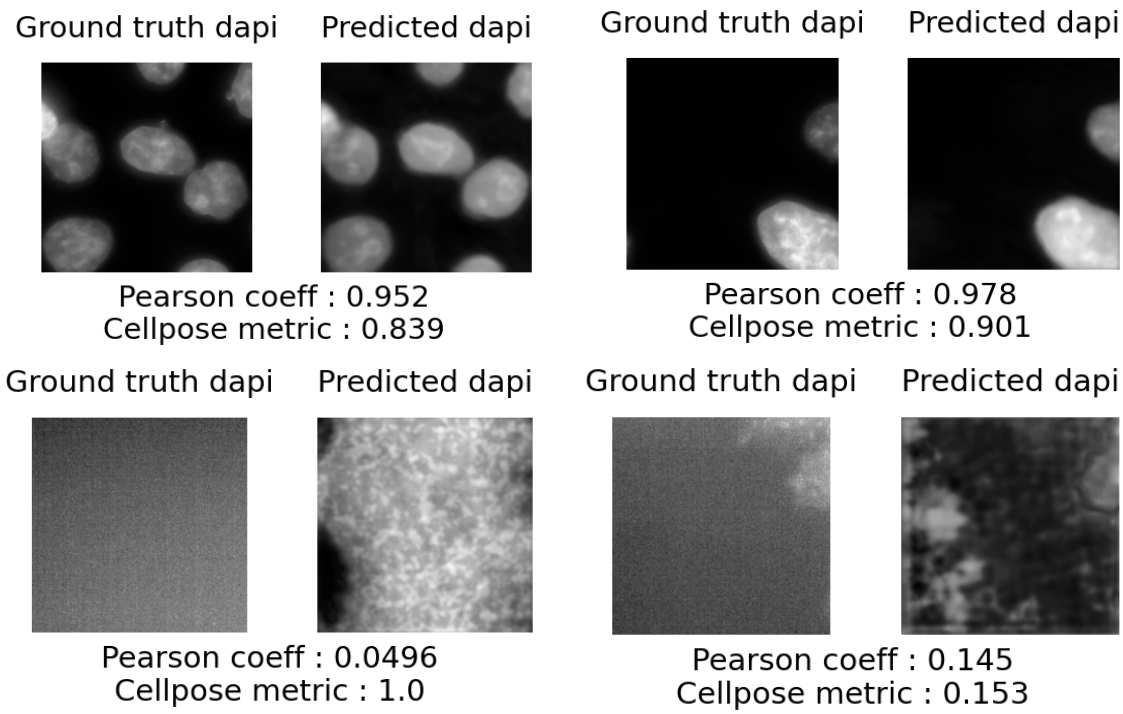


Figure 16: PCC and Jaccard Index (on Cellpose predictions) evaluated on empty crops and non-empty crops.

These changes did not induce much difference in our performance, the model still had trouble predicting empty crops. While we are confident that this issue can be solved relatively easily, we have decided not to focus on this aspect, as this problem clearly does not represent the main interest of the presented project. In addition, it can easily be bypassed by removing empty crops with classical methods prior to presenting them to the network. For these reasons, we focus our work on non-empty crops and the results we present in the following part are obtained by training our models on crops that contain at least one cell.

Specifically, our models are trained on 2626 non-empty crops and tested on 880 non-empty crops, during 150 epochs with early stopping. We train our models with a 3-channel input : two channels corresponding to two z-stacks extracted from the Brightfield image and one channel for the phase contrast image. As ground truth, we use the maximum intensity projection (MIP) of DAPI images. More precisely, we transform each 3D DAPI image into a 2D-image by selecting the maximum pixel value along each z-stack. This choice of input was selected thoroughly : we tested different input configurations, observed their impact on the model performance and chose the inputs that induced the higher Pearson correlation coefficient (PCC).

	Brightfield	Phase contrast	Brightfield + Phase contrast
train	0.94577	0.93035	0.95450
val	0.89449	0.90399	0.92744
test	0.90113	0.91090	0.93307

Table 1: Choice of input for our models based on the training of a U-Net model with L1 loss

	U-Net	U-Net on Steroids
train	0.95450	0.96613
val	0.92744	0.94532
test	0.93307	0.95203

Table 2: Mean Pearson correlation coefficient on train, validation and test sets for U-Net and U-Net on steroids models trained with L1 loss.

We compare U-Net model with U-Net on steroids model whose encoder is a DenseNet121 model pre-trained on DenseNet121. Both models are trained three times with three different losses : L1 loss, Pix2pix loss and Perceptual loss. We observe in Table 2 and Figure 17 that U-Net on Steroids gives better results than U-Net.

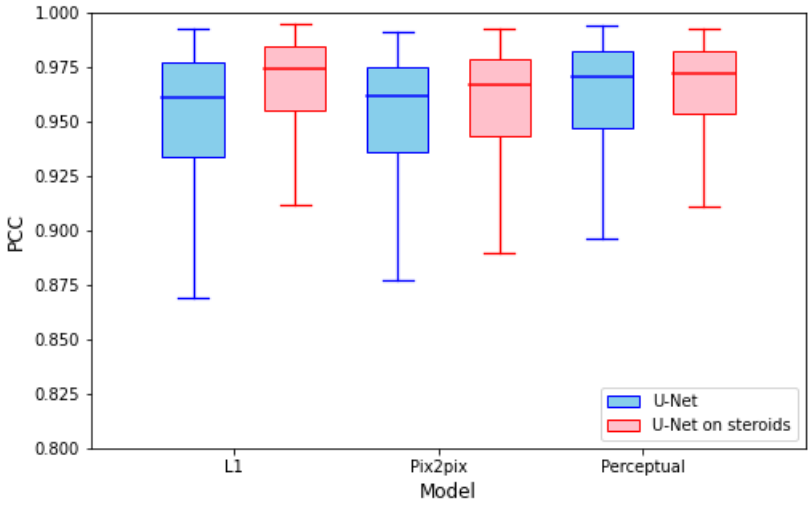


Figure 17: Pearson correlation coefficients using three different losses (L1, Pix2pix and Perceptual loss) for Dapi fluorescence prediction with both U-Net and U-Net on steroids, measured over 880 test crops.

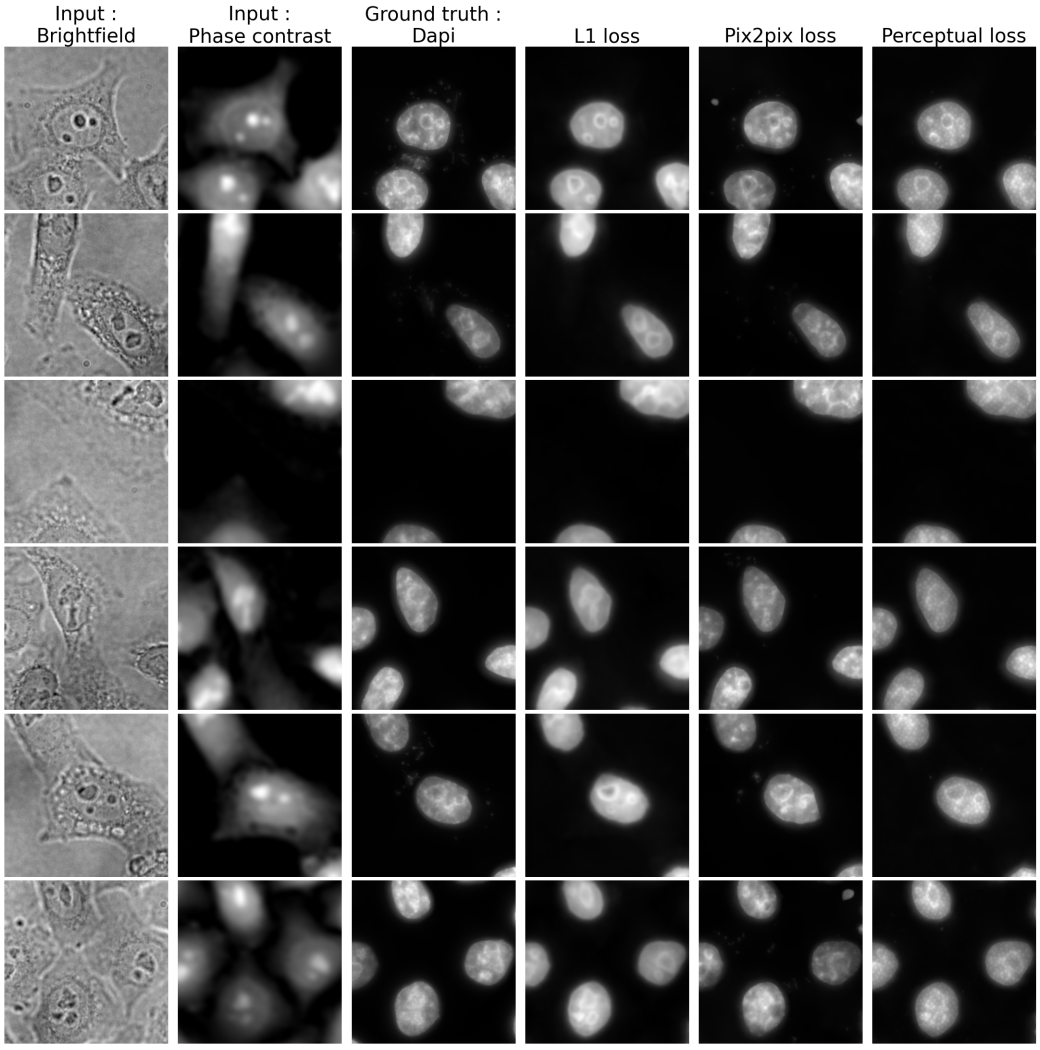


Figure 18: Comparison of different losses used for solving the In silico labeling task. We predict the DAPI fluorescence image from brightfield and phase contrast input images with U-Net on steroids model. From left to right : Brightfield input image, phase contrast input image, DAPI ground truth fluorescence image, the predicted image obtained by using L1 loss, the predicted image obtained with Pix2pix loss and the predicted image obtained with the Perceptual loss.

As expected, we observe in figure 18 that L1 loss produces blurry results whereas Pix2pix and Perceptual losses allow to generate higher quality images that are visually much closer to their associated ground truth image. Despite this visual observation, L1 model gives the highest mean PCC since it was trained to minimize an euclidean distance between the pixels. On the other hand, for a perceptual metric, the model trained with content loss would naturally obtain better results since it was trained to capture perceptual similarities between the images. We can observe that the model is pretty good at predicting shape and localization of nuclei within the fluorescence image. However, it seems to have trouble in learning the fluorescence quantity information. More precisely, it allocates more or less the same "amount" of fluorescence to each cell, or in a random manner. Here, DAPI fluorescence allows to quantify the DNA housed in the nucleus of each cell. Unfortunately, this can already be an indicator of the difficulty of our next task that consists in predicting cell cycle stages. Predicting the fluorescent labeling of cell cycle stages is different from predicting subcellular structures such as nuclei. The goal is to label the belonging to a phase, which highly relies on applying fluorescence where it counts.

3.5.2 Fluorescence prediction from DIC images

Here, all of our models are trained on 600 crops with a 3 channel-image as input corresponding to 3 different z-stacks of our DIC image and evaluated on 200 crops of test. In this section, we predict the Hoechst image that corresponds to the fluorescence labeling of the cells nuclei as well as the Cy5 image that labels the cells membrane. We also predict the GFP and Cy3 images that correspond to the fluorescence labeling of the cells cycle. Note that here, it is less likely to encounter empty crops since the images resolution is smaller.

Prediction of the Hoechst and Cy5 channels

In this part, we predict the Hoechst channel from DIC images and compare the three losses (L1, Pix2pix and Perceptual loss) the same way we did for DAPI channel in the previous section. Like DAPI images, the goal is to label the nuclei of the cells. Therefore, our observations are rather similar as before. We also try to predict Cy5 image that labels the cells membrane. As we can see in figure 19, our model trained with Perceptual loss gives very good results on Cy5 images with a mean PCC of 0.9543 on the test set. The model is able to predict the cells membrane and therefore to provide a fluorescent mask for each cell of the image. In figure 21, we observe the prediction of Hoechst images for the different losses. This confirms the observations we did in the previous section : perceptual and pix2pix losses generate less blury and more realistic images. Moreover, we notice the same problem as before with DAPI images : the model fails at applying the right amount of fluorescence, especially with L1 model. Perceptual model allows to alleviate this problem but it still does not match the ground truth fluorescent labeling. However, we still obtain good predictions that generate the right shape and localization of each nucleus.

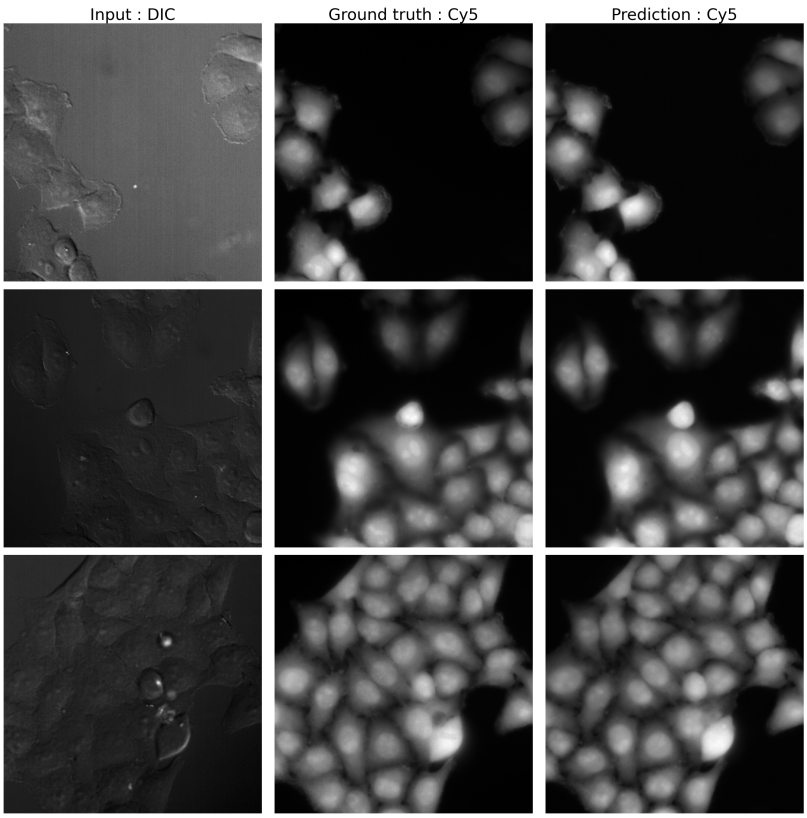


Figure 19: In silico labeling for Cy5 prediction from DIC images : Perceptual loss

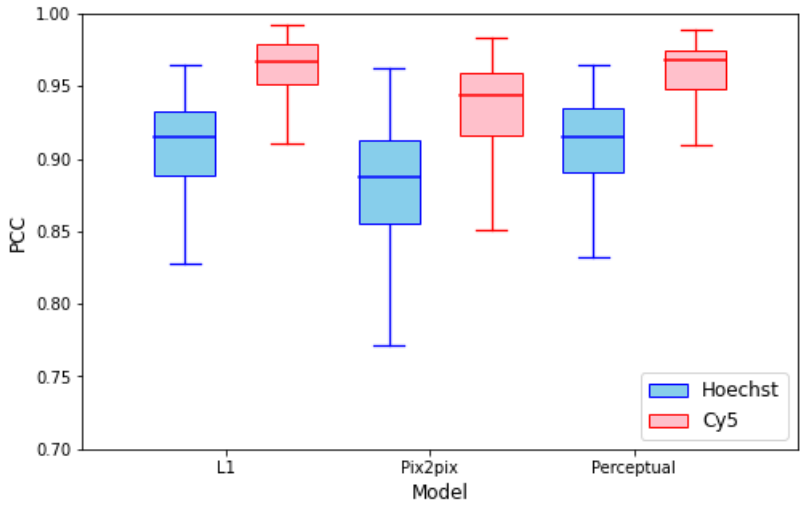


Figure 20: Hoechst and Cy5 prediction from DIC images : PCC evaluated on the test set from U-Net on steroids model.

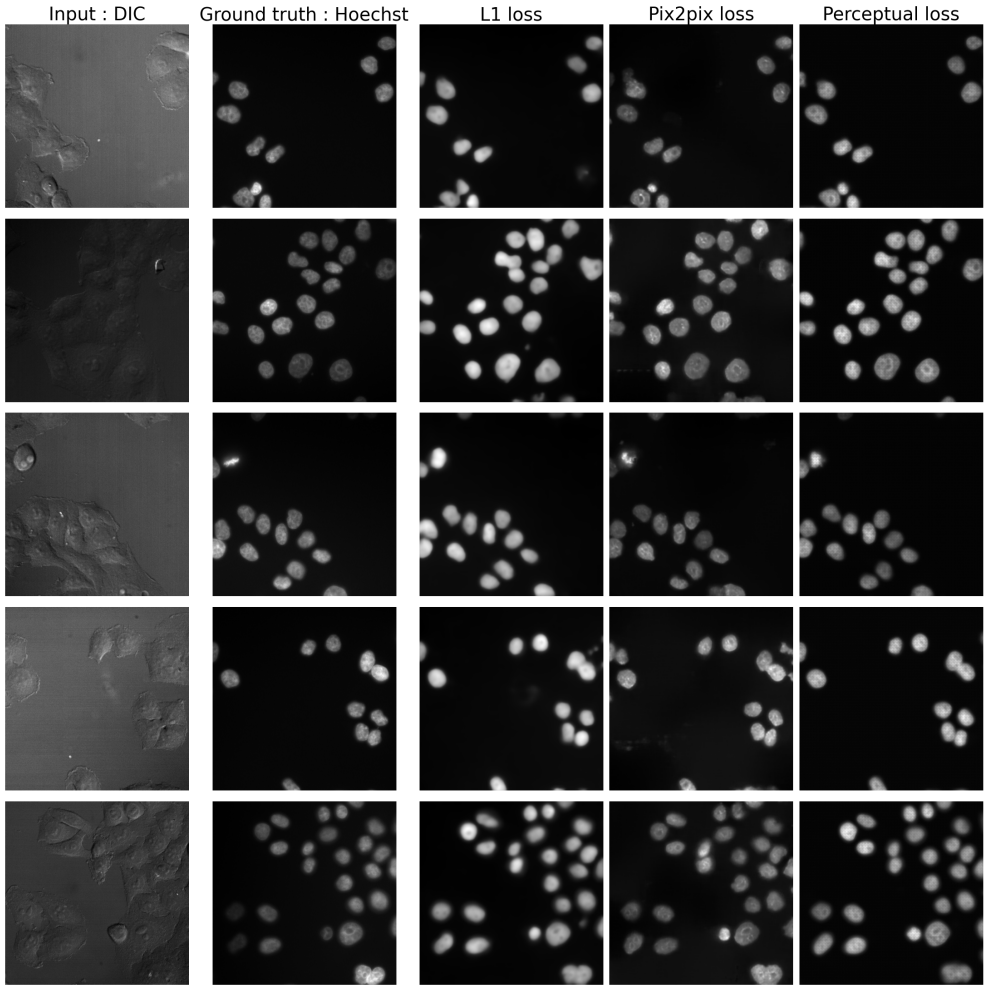


Figure 21: In silico labeling for Hoechst prediction from DIC images : different losses

Prediction of the cell cycle

The cell cycle is the process a cell undertakes to replicate its information and to divide into two identical cells. The cell cycle is composed of interphase (G1, S, and G2 phases), followed by the mitotic phase (mitosis and cytokinesis). During interphase, the cell grows and replicates its DNA. During the mitotic (M) phase, the cell separates its DNA into two sets and divides its cytoplasm, forming two new cells. At the end of each go-round, the two daughter cells can start the exact same process over again from the beginning. In this part, we try to predict the fluorescence labeling of the cells that undergo interphase. Specifically, cells in G1 phase are labeled by the red fluorescent dye Cy3, and those in S/G2 phases by the GFP green fluorescent molecule. Our goal is to try using In Silico Labeling to predict these Cy3 and GFP fluorescence images.

We train our models to predict both GFP and Cy3 channels together. Specifically, our models are trained with a 3-channel input, corresponding to three z-stacks extracted from the DIC image. As ground truth, we use a 2-channel image : one channel corresponding to the maximum intensity projection (MIP) of the Cy3 image and the other channel corresponding to the MIP of the GFP image. In this part, we train a U-Net on steroids model with the three different losses we studied, and compare results on predicting Cy3 and GFP separately.

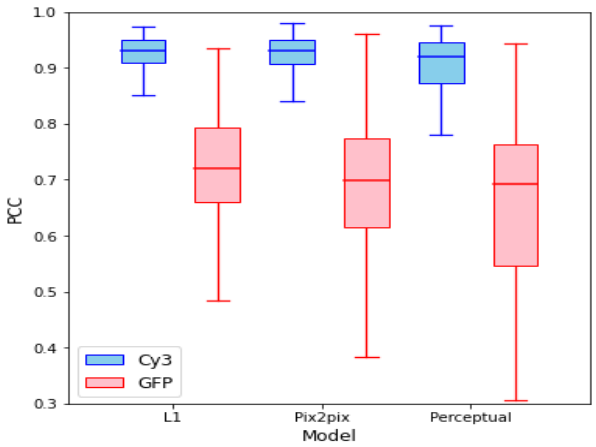
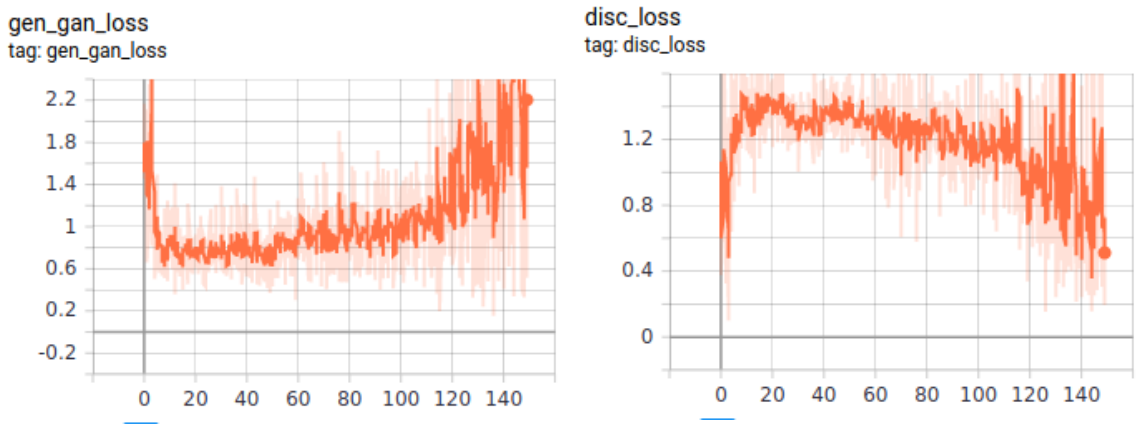


Figure 22: Cell cycle prediction from DIC images : PCC on the test set from U-Net on steroids model. We observe that results obtained on GFP predictions are markedly lower than those obtained on Cy3, and that they suffer from a higher variance.

In figure 22, we observe good results for the Cy3 images prediction with a mean pearson correlation coefficient of 0.9203 and standard deviation of 0.04073 evaluated on the test set. However, the model has difficulty in predicting the GFP channel, resulting in a rather low mean PCC of 0.7211 with a standard deviation of 0.0936. As we suspected, we observe in figure 24 that the model fails to apply fluorescence where it counts. This results in a GFP image that tends to label every cell with fluorescence instead of only labeling the cells that actually belong to S/G2 phase. We also notice this problem in Cy3 generated images where all cells have high illumination. These results are rather discouraging about In Silico Labeling being a suitable method for predicting stages of the cell cycle.



(a) Generator loss

(b) Discriminator loss

Figure 23: Pix2pix model learning curve. Adversarial training involves that generator and discriminator learning curves are symmetric.

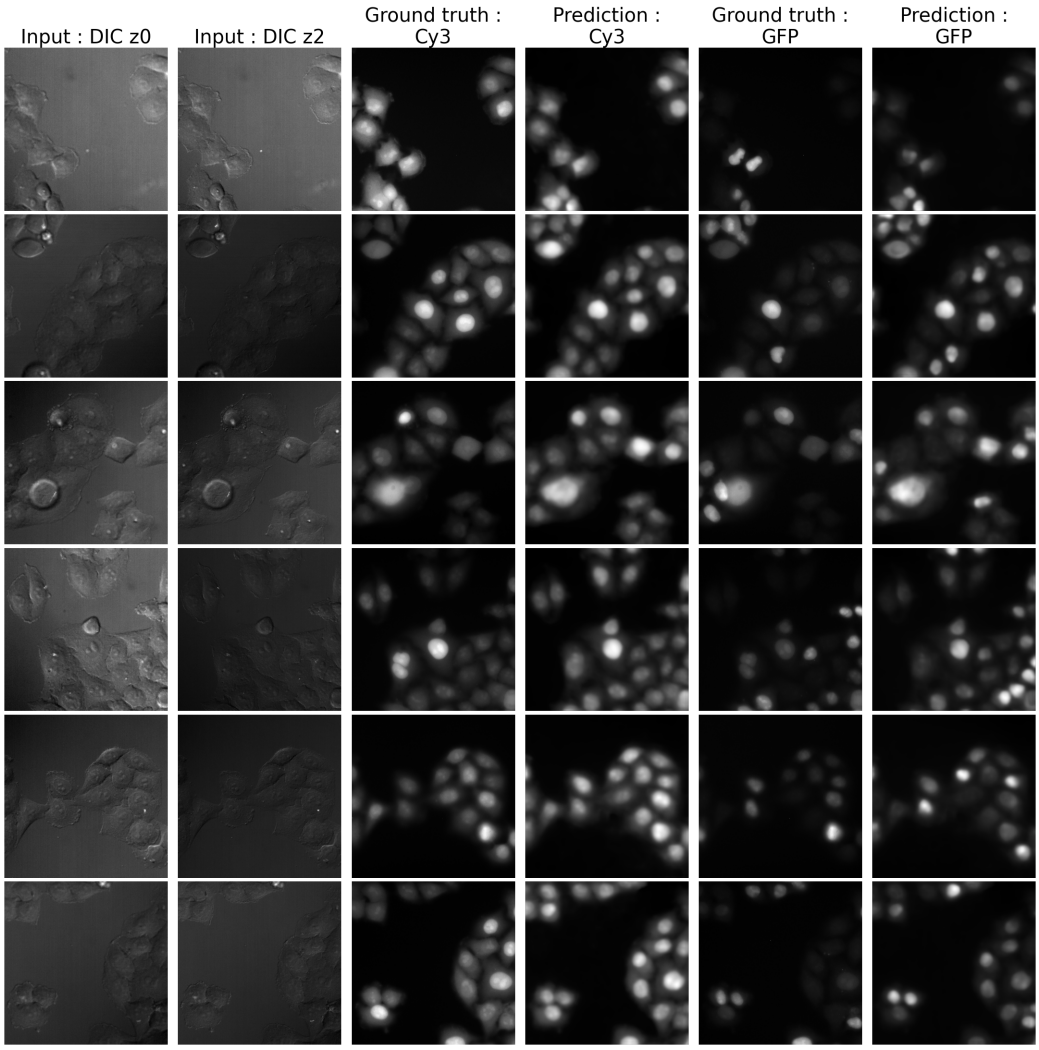


Figure 24: In silico labeling for cell cycle prediction from DIC images : L1 loss

4 Application of ISL to nuclei segmentation

Throughout this section, we will investigate the use of in silico labeling for pretraining segmentation networks. We will show that ISL can be more than predicting a fluorescent image and be useful for other biological related tasks, such as nuclei segmentation, through transfer learning. Of note, this strategy is general and can in principle be applied to segmentation of other cellular compartments, as long as they can be labeled.

4.1 Nuclei semantic segmentation from transmitted light images

In this part, we try to train a semantic segmentation model to predict cells nuclei masks directly from our transmitted light images (brightfield and phase contrast). The segmentation masks we used as ground truth were obtained by applying CellPose pre-trained model to our DAPI images. The model we used here is a U-Net on steroids with a sigmoid activation in the last layer in order to output, for each pixel, its probability of belonging to a nucleus. Specifically, our prediction output shape matches the input's spatial resolution (width and height) with only one channel that consists of a binary mask to indicate for each pixel if it belongs to a nucleus or not.

In order to measure the performance of our semantic segmentation model, we first thought of the pixel accuracy that simply reports the percent of pixels in the image that were correctly classified. However, this metric is known to provide misleading results when the class representation is small within the image, as it will mainly report how well the model identifies negative case (ie. where the class is not present). This is specifically our case since most of our crops contain very few nuclei or no nuclei at all.
Therefore, we thought of the Intersection Over Union (IOU) metric, also known as the Jaccard Index, which quantifies the percent overlap between the target mask and our prediction output. Specifically, the IoU metric measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across both masks. Note that technically the IoU is not defined if there are no true or predicted labels, which here corresponds to the case where both our target and prediction masks are empty. For our task, this actually corresponds to a good prediction since our model was able to detect that the image contains no nucleus at all. Therefore we modified our IoU metric by setting it to 1.0 in this particular case.

$$IoU = \frac{|target \cap prediction|}{|target \cup prediction|} \tag{15}$$

The model we used is a U-Net on steroids trained on 512x512 crops based on 554 train images and tested on crops from 185 test images. We used brightfield and phase contrast images as input to predict the nuclei binary masks. The loss function we used is a Jaccard Distance that just corresponds to 1 - IoU, which allows to measure how dissimilar two sets are. We trained the model during 150 epochs with a batchsize of 8. We obtain our final prediction mask by transforming the output probabilites to binary values by applying a 0.5 threshold : each pixel with a probability above 0.5 is set to 1 (belonging to a nucleus) and each pixel with a probability under 0.5 is set to 0 (belonging to the background).
The results we obtained are very satisfying with a mean Jaccard Index of 0.92349 on the test set. In the next section, we prove that pre-training this segmentation model with a ISL model trained for fluorescence nuclei prediction task can drastically reduce the amount of training data required.

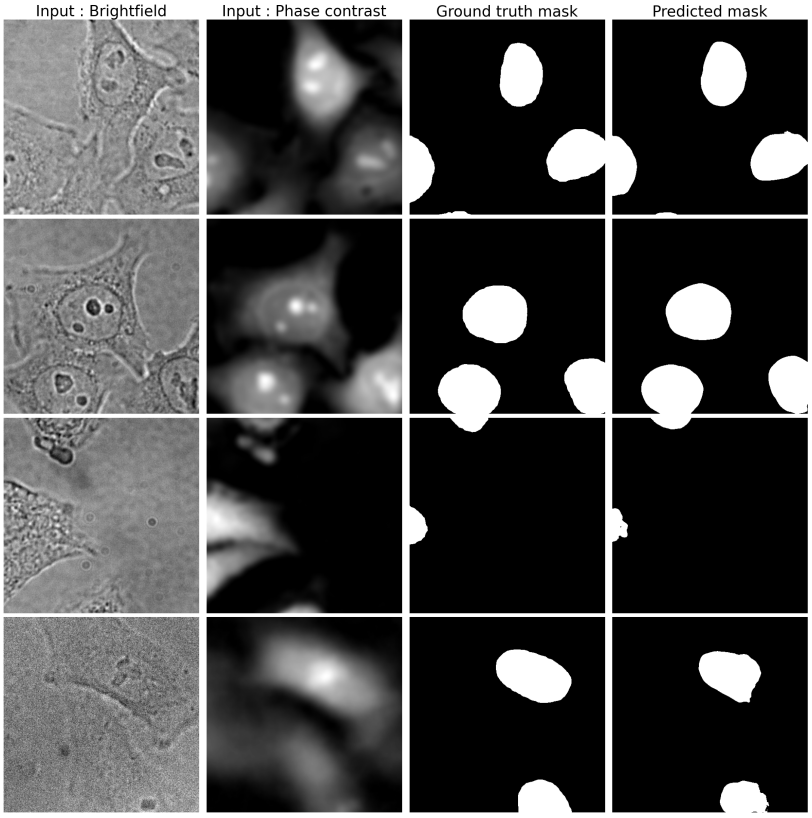


Figure 25: Nuclei semantic segmentation from Brightfield and phase contrast images on 512x512 test crops

4.2 In silico labeling as a pretext task

4.2.1 Pretext tasks

To give a sense of what we are doing next, we introduce the concept of downstream and pretext tasks. Downstream tasks are tasks with insufficient annotated data samples, for example classification or detection tasks for which getting annotation can be time-consuming and expensive. In order to help their training despite the lack of annotations, we commonly use pretext tasks. Pretext tasks are pre-designed tasks that aim at learning feature representations that can be easily adaptable and useful for downstream tasks thanks to transfer learning for example.

Generally, pretext tasks are tasks that involve self-supervised learning. Self-supervised learning is a form of unsupervised learning where the data itself provides the supervision. In general, some part of the data or of its properties is "withheld", and a network is trained to predict the "missing" information. Specifically, we create "artificially" our own task and labels from the data we have, and train a model to solve it in order to extract learned features that can then be used to solve a downstream task. For example, image colorization is a type of self-supervised pretext task : we train a model to colorize images, i.e to predict pixel colour from a monochrome input. In order to solve this task, the model is forced to learn spatial and semantic information about the image. We can also train a network to learn how to rotate an image (image transformation) or to fill in blanked out regions of an image based on surrounding pixels (image inpainting), or simply to reproduce the input image (autoencoder). In NLP, Word2Vec is based on a self-supervised pretext task that consists in learning word representations by predicting a missing word from an input sentence.

We prove that In silico labeling (ISL) can be very useful for nuclei segmentation by using it as a pretext task. Here, it is not a pretext task based on self-supervision learning but the goal is the same. We prove that pre-training our segmentation model on ISL can allow to reduce drastically the number of annotated data required. For this nuclei segmentation task in particular, we use the ISL model that was trained to predict nuclei fluorescence images, ie dapi images. More precisely, we use the weights learnt during its training as an initialization for our segmentation model. We compare the performances of this pre-trained segmentation model with a basic segmentation model, for different numbers of training images.

4.2.2 Results

This part consists in comparing two models : one "basic" segmentation model and one pre-trained segmentation model. Our "basic" segmentation model uses the same exact model framework as in section 4.1. For our pre-trained segmentation model, we select the ISL model trained to predict nuclei fluorescence images, ie dapi images, with the highest pearson coefficient. Specifically, we select the U-Net on steroids model trained for ISL task with a L1 loss (see section 3.5.1) and use its weights to initialize our semantic segmentation model. We train each model on different numbers of images and compare their performance on the same test set that consists in 185 images. Both models use phase contrast and brightfield images as input to predict the nuclei binary masks in output, and are trained using the Jaccard Distance, like in section 4.1.

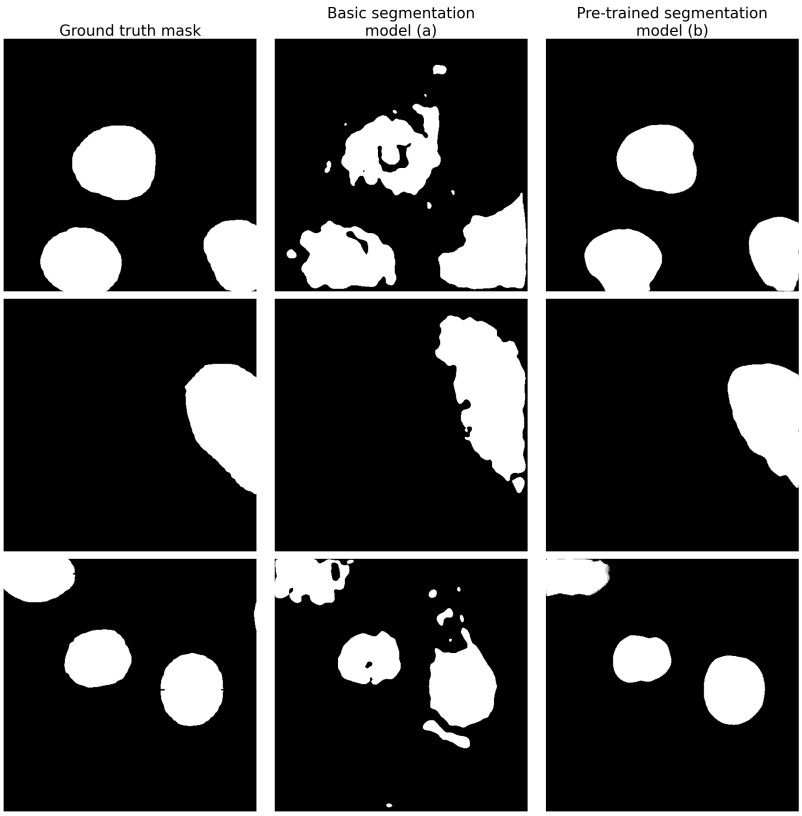


Figure 26: Results on semantic segmentation : training on **1** image. **(a)** Basic segmentation and **(b)** segmentation pre-trained on ISL.

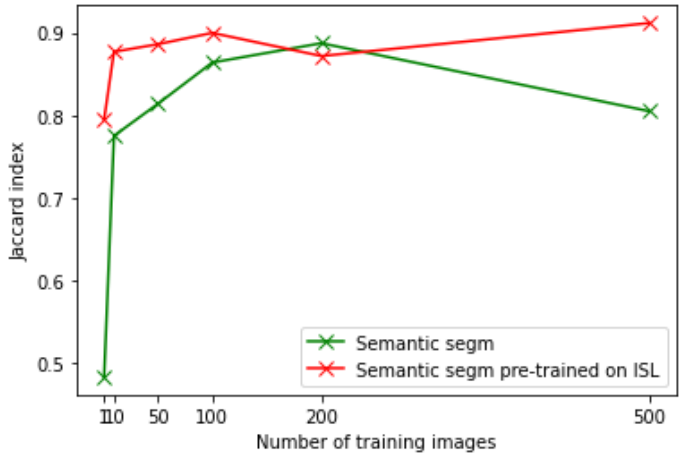


Figure 27: Evolution of the jaccard metric on the test set in function of the number of training images.

We observe very interesting and encouraging results for this part. Figure 27 shows that globally, our pre-trained model performs better than the basic semantic segmentation model. But more interestingly, we notice that our pre-trained segmentation model performs very well on the test set even for very few training images. Trained on 1 image only, it already reaches a Jaccard index of 0.79490 while the "basic" model gives a Jaccard index of 0.48235. The difference is also easily observable for 10 and 50 training images. We also illustrate this in figure 26 where we can compare the nuclei masks output by each model trained on 1 image. We observe that the pre-trained model produces segmentation masks much closer to the ground truths than the basic model. This proves that using In Silico Labeling as a pretext task for nuclei segmentation can give very good results and significantly reduce the number of training data required. This is very useful since obtaining anotated ground truth segmentation masks can be complicated and time consuming. Furthermore, a smaller number of training images naturally allows a faster training.

These results can be explained by the fact that predicting fluorescent nuclei images is quite similar to performing nuclei segmentation task, therefore we only need very few annotated data to fine-tune the pre-trained model. Indeed, to generate nuclei fluorescence images, ISL model needs to extract features from the input image that are similar to the ones needed to perform nuclei segmentation. Amongst other, ISL model has to learn shape and localization of the nuclei, which are obviously useful features for nuclei segmentation as well. Although we applied this to nuclei segmentation, note that this strategy is general and can be applied to segmentation of other cellular compartments, as long as they can be labeled and are suitable to ISL method.

5 Conclusion

In Silico Labeling was developed in 2018 as a deep-learning approach that aims at generating fluorescence microscopy images from transmitted-light images of unlabeled biological samples. This method represents a groundbreaking tool for biological research.

During this internship, we developed our own ISL model and applied it to our data. The model performed very well on predicting subcellular structures such as cells membrane or nuclei. This work has enabled us to study and compare state of the art methods that aim at solving image-to-image translation task. Specifically, we compared different training losses for our model such as Pix2pix loss that makes use of Conditional Generative Adversarial Networks, and Perceptual loss that relies on transfer learning and features extraction from pre-trained models.

While ISL has been proved to perform well on subcellular compartments, it is not very clear which proteins are suitable or not for this kind of approach. In particular, our work revealed that ISL method seems to have difficulty to predict cell cycle stages. The model fails at applying fluorescence where it counts and therefore fails to label a cell with either Cy3 (phase G1) or GFP (S or G2 phases). However, other approaches we didn't have time to try could be the object of future work and research. For example, we could think about optimizing our model and choose an architecture more complex than U-Net, we could also question the use of DIC images as input for our model.

Finally, we have proved that ISL can be more than predicting a fluorescence image and can be particularly useful for segmentation tasks. Our results for this part are very satisfying. We prove that using ISL model on DAPI images to pre-train a nuclei segmentation network can allow to drastically reduce the number of annotated data required. The pre-trained segmentation network only needs few data to achieve a good performance. We took the example of nuclei segmentation but this strategy can in principle be applied to segmentation of other cellular compartments, as long as they can be labeled.

References

[1] Eric M. Christiansen et al. "In Silico Labeling: Predicting Fluorescent Labels in Unlabeled Images". In: *Cell* 173.3 (2018), 792–803.e19. ISSN: 0092-8674. DOI: https://doi.org/10.1016/j.cell.2018.03.040. URL: http://www.sciencedirect.com/science/article/pii/S0092867418303647.

[2] Martin Weigert et al. "Content-Aware Image Restoration: Pushing the Limits of Fluorescence Microscopy". In: *bioRxiv* (2017). DOI: 10.1101/236463. eprint: https://www.biorxiv.org/content/early/2017/12/19/236463.full.pdf. URL: https://www.biorxiv.org/content/early/2017/12/19/236463.

[3] Chawin Ounkomol et al. "Label-free prediction of three-dimensional fluorescence images from transmitted light microscopy". In: *bioRxiv* (2018).

[4] C. Belthangady and Loic A. Royer. "Applications , Promises , and Pitfalls of Deep Learning for Fluorescence Image Reconstruction". In: 2019.

[5] Dumitru Erhan et al. "Visualizing Higher-Layer Features of a Deep Network". In: *Technical Report, Univeristé de Montréal* (Jan. 2009).

[6] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[8] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[9] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[10] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[11] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[12] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[13] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[14] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[16] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[17] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[18] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[19] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[20] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV].

[21] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. *A Neural Algorithm of Artistic Style*. 2015. arXiv: 1508.06576 [cs.CV].

[22] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. 2016. arXiv: 1603.08155 [cs.CV].

[23] Carsen Stringer, Michalis Michaelos, and Marius Pachitariu. "Cellpose: a generalist algorithm for cellular segmentation". In: *bioRxiv* (2020). DOI: 10.1101/2020.02.02.931238. eprint: https://www.biorxiv.org/content/early/2020/02/03/2020.02.02.931238.full.pdf. URL: https://www.biorxiv.org/content/early/2020/02/03/2020.02.02.931238.