# U-Net: Segmentation of Biomedical Images

**Implementation and contributions to the U-Net network, a deep convolutional encoder-decoder trained to perform semantic segmentation on biomedical images, in order to both distinguish whether there is a disease and localize the abnormal area.**

**Flavio Giorgi[1], Filippo Liguori[2] & Valerio Mannucci[2]**
Professor: Emanuele Rodolà
Course: Deep Learning and Applied Artificial Intelligence

This report describes the work carried out as final project of the course *Deep Learning and Applied Artificial Intelligence*. We are going to introduce the semantic image segmentation problem and its importance in biomedical imaging, and describe the features and performance of UNet, the convolutional network we studied and implemented to address semantic segmentation tasks.

## Introduction

'*Our ability to measure and alter biology far surpasses our ability to actually understand it*', writes Ed Regis in his renowned book *What is life?*, citing Brendan Frey, co-founder of the Canadian company *Deep Genomics*. We indeed know very little about the complexity of biological processes, compared to the accuracy we have achieved in observing and even engineering them. Imaging, synthesis and sequencing techniques can be used in perspective to build complex genetic circuits that replicate pre-programmed metabolic processes [**?**], and to advance the diagnosis and treatment of many diseases poorly known today. State of the art is still far from this goal, but a lot of cutting edge research is being dedicated to integrating Engineering, Biology, Physics and Computer Science to better understand biological processes and analyze biomedical data.
After briefly introducing both the **segmentation problem** in general, with a focus on the segmentation of biomedical images and its relevance in the first section, in the second section of the review, we will give a deeper insight into our **work environment** and its features. In the third section, we will describe the **dataset** used to train the network, the pre-processing steps we performed to uniform and analyze it, and the purpose and features of the data augmentation we implemented. In the fourth section, we will describe the **architecture** of the network, characterizing all the layers. As already mentioned, our network is based on U-Net, a fully connected convolutional network developed at the Computer Science Department of the University of Freiburg, Germany, but we will stress our personal contributions. Before ultimately presenting the **performances** of the network on two different datasets and summarizing the conclusions and outlook of the work recapped in this report, in the fifth section we will delineate the features of the scheduling of optimizations parameters and tuning of the **hyperparameters**.

[1]Sapienza Università di Roma, Dipartimento di Informatica, Master Student
[2]Sapienza Università di Roma, Dipartimento di Fisica, Master Student

# 1 Theoretical Context

## 1.1 Segmentation Problem

The field of Computer Vision has largely benefited from the growing complexity of deep learning algorithms that have been developed in the last lustra. U-Net, whose architecture we have implemented, expanded and studied, exploits the potential of a convolutional encoder-decoder model to perform **semantic segmentation**. There are various levels of granularity in which the computers can gain an understanding of images. For each of these levels there is a problem defined in the Computer Vision domain. Starting from a *coarse grained* down to a more **fine grained** understanding, we can define the following list of segmentation problems, and describe each of them:

1. **Image Classification**
2. **Classification with Localization**
3. **Object Detection**
4. **Semantic Segmentation**
5. **Instance Segmentation**

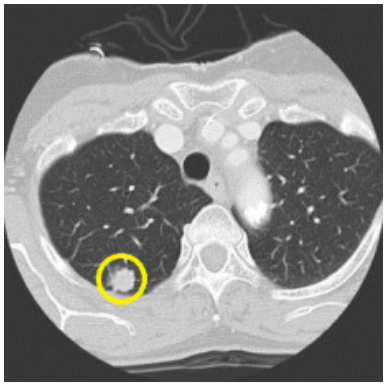
*Image classification* is the most fundamental problem in Computer Vision, in which the algorithm takes as input a single image containing only one object, and is expected to output a discrete label that most accurately describes the image. One more step is needed if the algorithm is expected to perform *classification with localization*, where along with classification the object should be localized in a frame of reference defined considering the image boundary. *Object detection* takes image classification to the next level. It can be performed on images that contain more than one object, and classification and localization must be obtained for all of them. Further sophistication is introduced by *semantic segmentation*, namely labelling each pixel of an image with a corresponding class of what is being represented. Unlike the previous ones, the output of a semantic segmentation algorithm is another image, typically the same size as the input, in which each pixel is classified to a particular class. The highest level of sophistication is obtained through *instance segmentation*, wherein along with semantic segmentation, the algorithm is expected to classify each instance of a class separately. For example, semantic segmentation of an image of a population of cells (being them cancer cells, differentiating stem cells or a culture in vitro), outputs an image the same size of the original, with the **masks** of the cells, as we will see in the next sections. Instance segmentation would further label each of the masks, for example outputting an image of the masks of the cells, each coloured in a different way, differentiating between the instances of the '*cells class*'.

## 1.2 Segmentation of biomedical images

Humans possess a great variety of sensory systems that diagnosticians can exploit to identify whether a certain medical disease is present in a patient or not. Automating this process in a robust and accountable way is still a very open challenge today. When a human observer views a scene, his or her visual system segments the scene semantically, and this

feature can be used widely for diagnostic purposes. Automated visual analysis of biomedical images can prove itself as an extremely powerful tool to speed up the diagnostic process. By marking the desired type of cells with a known dye, the cells nuclei can be spotted and photographed. Subsequently identifying the nuclei through segmentation, allows researchers to identify each individual cell in a sample, and by measuring how cells react to various treatments, we can understand the underlying biological processes at work.

Figure 1: Pulmonary nodule



In this section we briefly present as an example an image segmentation problem of great importance in biomedical data analysis, namely '*pulmonary nodule detection and segmentation for early diagnosis of lung cancer*'. **Lung nodules** are small masses of tissue in the lung, that appear as round, white shadows on a **computerized-tomography** scan (CT)[1], as we see in figure 1. Lung nodules are usually 5 to 30 millimetres in size. A larger lung nodule, such as one that's 30 millimeters or larger, is very likely to be cancerous, and segmentation of CT scans, as the one in figure 1 can identify these regions and automatically diagnose the presence of nodules. In section 6 we'll see another application of our semantic segmentation network in biomedical research. We're not going to focus on other applications of segmentation in this report, but its features are widely exploited in other relevant problems, such as autonomous vehicles driving, geological sensing and precision agriculture.

# 2 Work Environment

The network has been built on Google Colaboratory, a platform that enables us to execute code directly on the Cloud, whose interactivity is favoured by the *Jupyter Notebooks style*. The features of Colab that make it useful to build a convolutional network, are the *interaction with the Drive* and the *hardware* Google provides to Colab users. We can *mount* Google Drive on our Colab notebook, and all the documents saved in *My Drive* are mounted in a directory called *my-drive*, in the root of our file system. This feature proves to be useful when dealing with a large dataset to train the network.

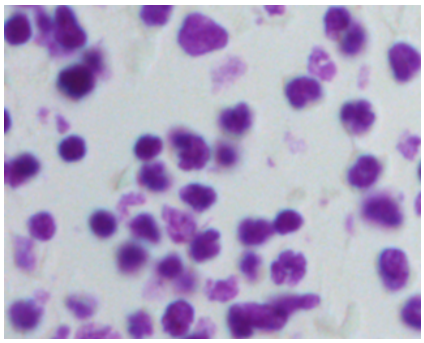
The second feature that made Colab essential in carrying out this project was the **Google GPUs**. We exploited the deep learning framework *PyTorch*, considered by many sources the leading one for research. There has been much recent research dedicated to studying the importance of the hardware on the training speed of deep neural networks, and some works by *Indigo AI* showed that GPUs can be 250 times faster than CPUs. Google has developed ulterior hardware in this context, releasing the *Application-Specific Integrated Circuit* **Tensor Processing Unit**, that goes beyond the GPU technology and was made available for third party use in 2018. The TPU was developed specifically for neural network machine learning, but is still too specific for Google's own TensorFlow software[2]. Despite these considerations, the GPU Cloud made available by Google to Colab users, was a powerful hardware for our purposes and has enabled us with enough speed in training our network.

[1] An imaging procedure that uses computer-processed combinations of many X-ray measurements taken from different angles to produce cross-sectional (**tomographic**) images (*virtual 'slices'*) of the organ.

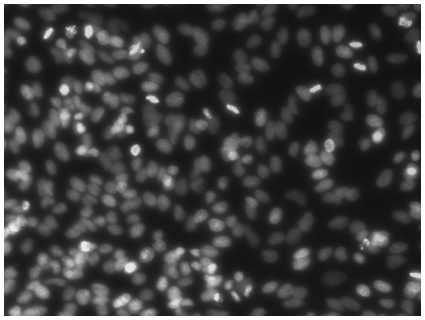
[2] Arguably the second leading framework for research in Deep Learning, and still the leading one in industry.

# 3 Dataset Analisys

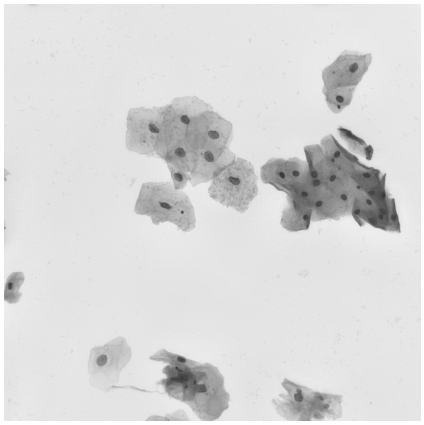
In order to train our network, we downloaded the nuclei images provided on the Kaggle website, as dataset for the *2018 Data Science Bowl: Find the nuclei in divergent images to advance medical discovery*[1]. As explained on the website in bibliography, the dataset contained 670 segmented nuclei images. Since the images were acquired under a variety of conditions, namely different tissues and cell type, magnification, and imaging modality (brightfield vs. fluorescence). The dataset diversity was designed purposely to challenge an algorithm's ability to generalize across these variations. Below, we give an example of the diversity of the images present in the dataset.



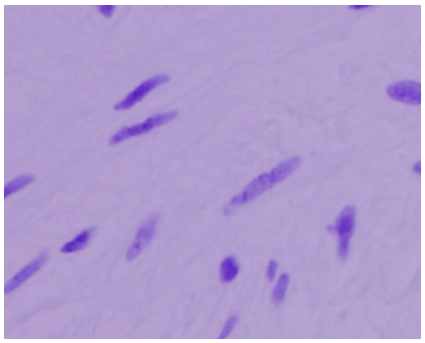
(a) Instance 1 of 2018 DSB



(b) Instance 2 of 2018 DSB



(c) Instance 3 of 2018 DSB



(d) Instance 4 of 2018 DSB

As mentioned, the dataset contained 670 couples of images, representing the *instance files* and the corresponding *mask*, or target image, namely the **ground truth** of the segmentation problem. The mask can be obtained by manual or automatic segmentation, but is supposed to be a very reliable segmentation of the instance. The first preprocessing step we had to perform on the dataset was **standardization**. The shapes of the images, being taken in different environment, were very variable. We had to uniform their shape to the optimal for our network, that we identified as $256 \times 256$. We had to act differently on the images bigger or smaller than the desired dimension.

- Images bigger than $256 \times 256$ were sliced;
- Images smaller than $256 \times 256$ were simmetrically padded to reach the desired dimension.

## 3.1 Data Augmentation

Convolutional neural networks have been subject of research and developement for almost 40 years, but their popularity has not risen until lately due to the limited size of available datasets to train them. In June 2017, Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton published the breakthrough *ImageNet Classification with Deep Convolutional Neural Networks*, describing 'supervised training of a large network with 8 layers and millions of parameters on the ImageNet dataset with 1 million training images.' Since then, even larger and deeper networks have been trained, due to even larger datesets. We know that the dataset is therefore essential in the learning process, and we performed data augmentation on the data presented in the last section, for two main reasons:

- First, there is very little training data available for the nuclei segmentation task, and with augmentation transformations we can obtain a lot of pictures from a single one;
- Elastic deformations allow the network to learn invariance to such deformations, without the need to see these transformations in the *image corpus*. This is particularly important in biomedical segmentation, since realistic deformations of the soft biological matter can be simulated efficiently.

Data augmentation is essential to teach the network the desired invariance and robustness properties, when only few training samples are available. In case of microscopical images especially random **elastic deformations** of the training samples seem to be the key to train a segmentation network with very few annotated images, as shown in [4].
We defined the method `transform()`, and randomly applied various transformations, comprising **horizontal** and **vertical flips**, **random rotations** and **regions erasing**, alongside **elastic transformations**[3], in order to augment the dataset and many more pictures, starting from the initial 670. Every time the model is trained, the dataset is therefore changed.

[3] Applying only elastic deformation would have made the data augmentation process too slow.

# 4 Architecture of the network

The architecture we inspired to is shown below. U-net is one of the most powerful architectures to perform biomedical images segmentation. We use it as a blueprint to build our own architecture exploiting the u-shape and the concatenation between encoder and decoder features.

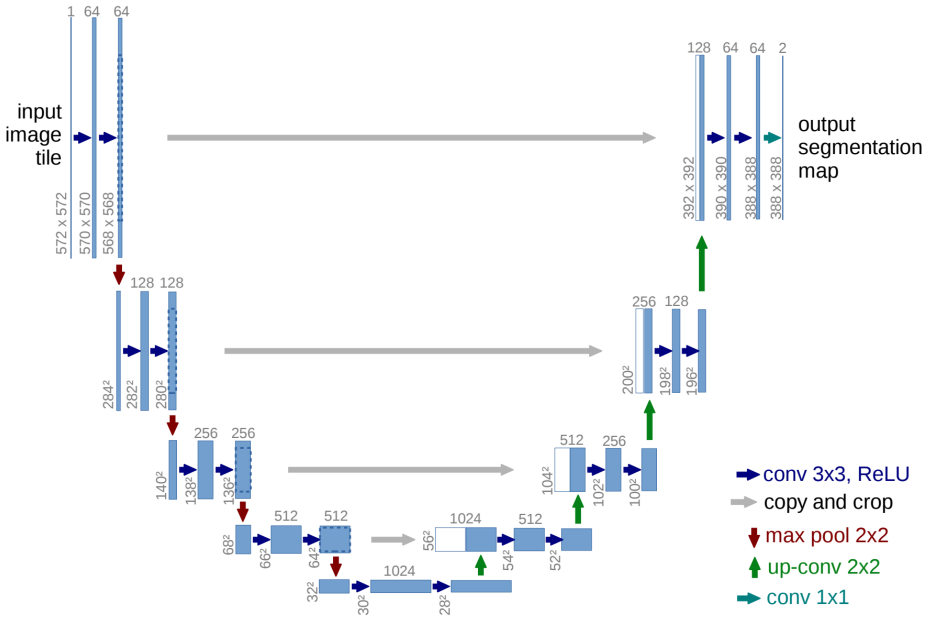



Figure 2: Architecture of the U-Net Network

In particular, we chose for the contraction path a network used to perform images classification: the VGG-16_bn network. We decided to use it due to the possibility to download pre-trained weights and to switch on and off the batch-norm layers. We performed two modifications to adapt VGG to our needs. We removed the fully connected layers and the softmax layer. Moreover, we slightly changed the width and height of the images.

As we can see, it consists of a **contracting part** (on the left) and an **expansive part** (right side). The contracting part consists of two consecutive $3 \times 3$ unpadded convolution layers, each *activated* by a rectified linear unit (**ReLU**), followed by a $2 \times 2$ max pooling operation with stride 2. These layers achieve **downsampling**. At each downsampling step the number of feature channels [3] is doubled. Every step in the expansive part consists of an upsampling of the feature map followed by a $2 \times 2$ convolution (denoted in [4] as "**up-convolution**") that halves the number of feature channels. Each feature map of the expansive part is concatenated with the corresponding feature map (**cropped**) of the contractive part. Each concatenation is followed by two $3 \times 3$ convolutions, each again activated by a ReLU. The cropping in the concatenation step is necessary because border pixels are lost in every convolution layer. To obtain the output segmentation map, a final $1 \times 1$ convolution is used to map each **feature map** to the desired number of classes. We will better analyze these variables in the next subsection, when discussing the *loss function* computation upon training. One of the changes we applied to the architecture shown in figure 3 was to add 4 convolutional layers, two after the last downsampling and two before the first upsampling.

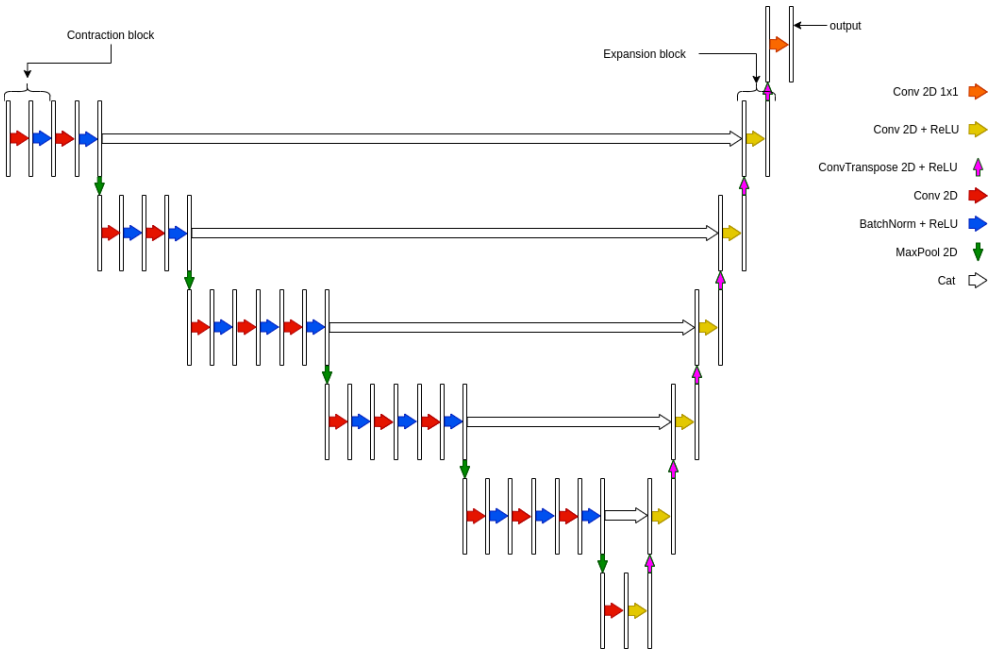



Figure 3: Our model architecture

The network has therefore a total of 27 convolutional layers. We thought of adding some **dropout** layers, but since our model is not prone to overfitting, we discarded this idea. Our model has a couple of implementation options that can be set before training. First, the user can 'switch on' the **batch normalization**, that as we know makes every training example interact with the other examples of the minibatch. Despite beneficial properties of batch-norm, including the enhanced stability of gradient descent and speed of training, we observed a worsening in the performance of the network, due to decreased contrast particularly at the border of the pictures. As common when making deep learning models available to third party users, our model can be used as *pretrained*. Its weights can then be frozen or it could be further trained to achieve better performance. The *contracting part* of our model, is actually the encoder **vgg16b**, that we have used just this way, importing the pretrained model. Actually we removed the linear layers from the said model, because it had been trained to perform classification on the ImageNet dataset, consisting of images belonging to 1000 different classes. The linear layers were therefore added for this specific purpose, and were not useful to achieve semantic segmentation, whose output is an image just like the input.

## 4.1 Training: loss and variables

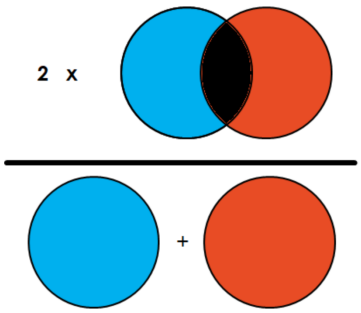
The first step in building the trainer, and therefore solve for the parameters $\Theta_i$ that best approximate the Multi-Layer Perceptron's function, is computing the loss. In our network, the loss function is computed by a **pixel-wise binary cross entropy with logit** :

$$\mathcal{L}_p(y) = -\frac{1}{N}\sum_{i=1}^{N} y_i log(\sigma(x_i)) + (1-y_i) log(1-\sigma(x_i)) \tag{1}$$

where $y_i$ represents the value of the i-th pixel of the ground truth mask, $\sigma(x_i)$ is the sigmoid function applied to the output of the model.

This is a different approach compared with the authors loss because they use a two channel output model and of course the soft-max loss. Before identifying the BCE as the optimal loss function, we experimented training with `DiceLoss`, simply implemented as the ratio of twice the overlap of the generated mask and the original, over the sum of the two masks, as shown in picture 4. We discarded due to the late convergence, and because the predictions of the masks looked less reliable than those obtained with BCE. We quantified the reliability of the predictions by using the library `sklearn.metrics` function `f1_score`. It is defined exactly like the Dice Loss; it outputs a real number between 0 and 1, where 1 quantifies the highest possible accuracy. The code we wrote to declare the trainer is:

Figure 4: Visualization of the Dice Loss



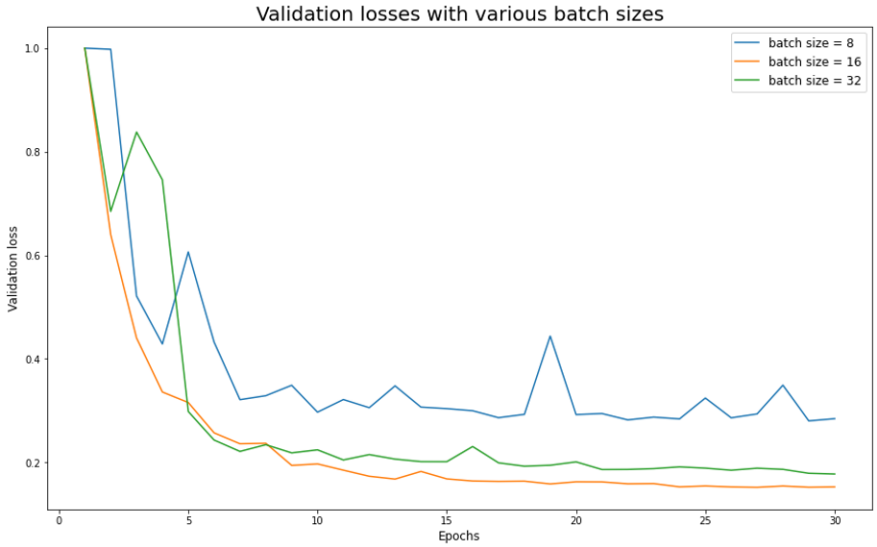

```
t = Trainer(optimizer = optimizer,
loss = loss,
model = model,
train_dataloader = train_dataloader,
test_dataloader=test_dataloader,
epochs=epochs,
device = device,
scheduler = scheduler)
```

where we used our implemented class `Trainer`, whose objects are:

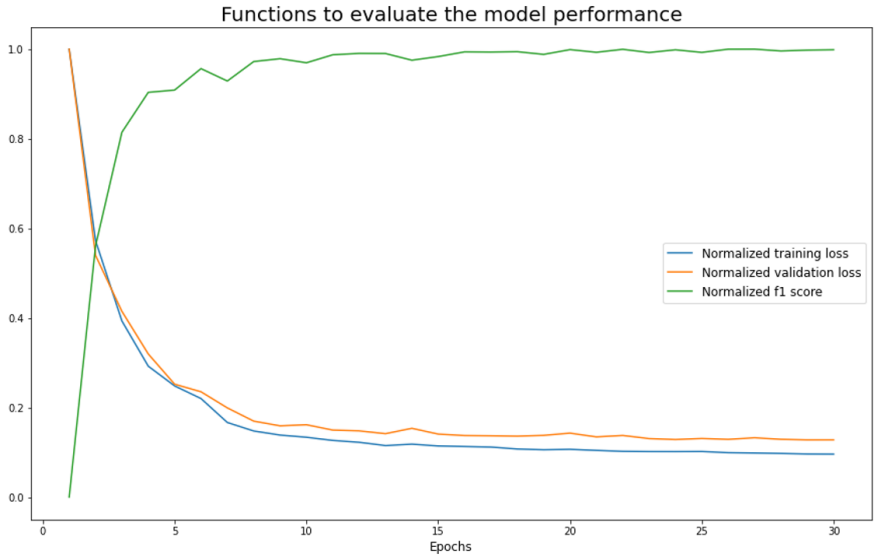
- `optimizer`: algorithm to perform **backpropagation**, in our case *SDG*;
- `loss`: loss function of the model;
- `model`: all the layers of the deep network described in section 4;
- `train_dataloader`: loader of the training set of images;
- `test_dataloader`: loader of the validation set of images, to spot eventual overfitting;
- `epochs`: number of training epochs;
- `device`: hardware supporting the training;
- `scheduler`: **learning rate scheduler**, that could be set as *Cyclic* or *WarmRestart*, as will be better explained in the next section.

# 5 Optimal hyperparameters tuning

After building the architecture of the network, the next step was optimizing the learning hyperparameters, namely the so called **linesearch algorithm**. We imposed two different schedulers to the **learning rate**, *WarmRestart* and *Cyclic*, in order to find the optimal parameters for the gradient descent algorithm. We split the dataset in training and validation sets, and performed training for 30 epochs, using various networks with different schedulers, and varying the *maximum learning rate* of the scheduler. As suggested in [4], we used a high **momentum** (0.9) such that a large number of the previously seen training samples determine the update in the current optimization step. We show below the loss function during validation and training, and the f1 score as a function of the epochs, for the optimal network.




(a)




(b)

As we see from the graphics, the optimal network, namely those that reach the minimum

of the loss within the least epochs, has a mini batch size of 16. We observed a better performance with scheduler *WarmRestart*, with maximum learning rate 0.4. We used it to solve a novel biomedical segmentation problem, as described in section 6.

# 6 Performance of the network: Results

Once we identified the optimal neural network, we used it to segment different datasets from the one we used to train the network. We present in this section two examples of the performance of the network on these datasets.

## 6.1 Differentiating neural stem cells

As a first application of our network, we segmented an image provided by the laboratory of *IIT@Sapienza* Center for Life Nano Science. The picture was taken with an epifluorescent microscope, and showed a population of fluorescing Neural Stem Cells differentiating and evolving *in vitro*. The dimensions of the image were $4704 \times 6240$. As shown in figure 5, the first step was slicing it in order to eliminate the borders on the bottom and on the right, owing to the acquisition method.

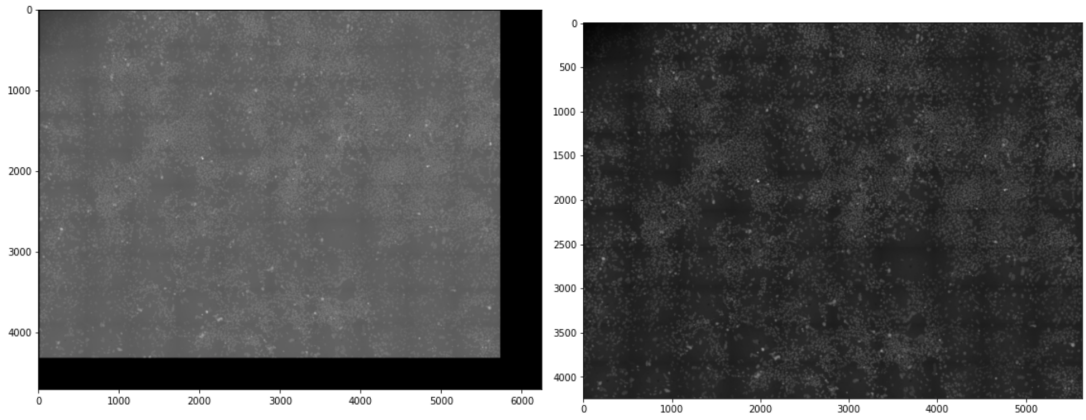
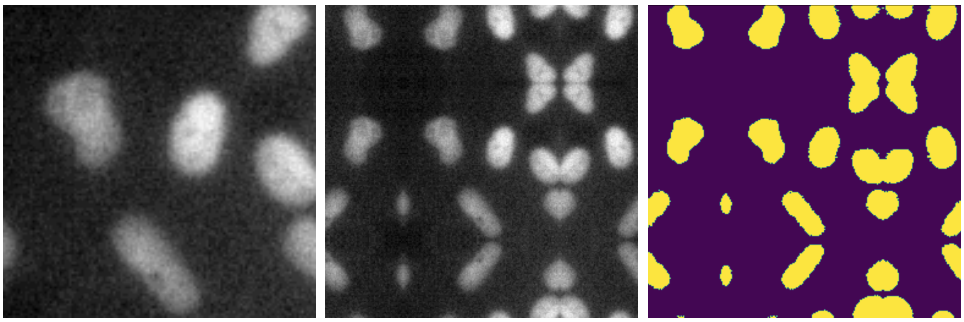



Figure 5: On the left the original image, on the right the cropped one

Since the neural network can only perform segmentation on images whose dimensions are $256 \times 256$, we had to slice the big picture on the right of figure 5 and then give the slices as input to the network. To avoid major *border* segmentation errors, we split the big picture in smaller tiles of dimension $128 \times 128$, applied **reflect padding** to them and obtaining $256 \times 256$ images, and ultimately sliced the inner $128 \times 128$ of the masks obtained as output of the network.



(a) Original tile (b) Padded tile (c) Segmented image

Figure 6: Process of segmenting the image on the right of figure 5

We could finally obtain the segmentation of the image on the right of figure 5, by merging (in order) all the unpadded masks, obtained by slicing the masks obtained just like the one on the right of figure 8. The result of this segmentation problem is shown below.

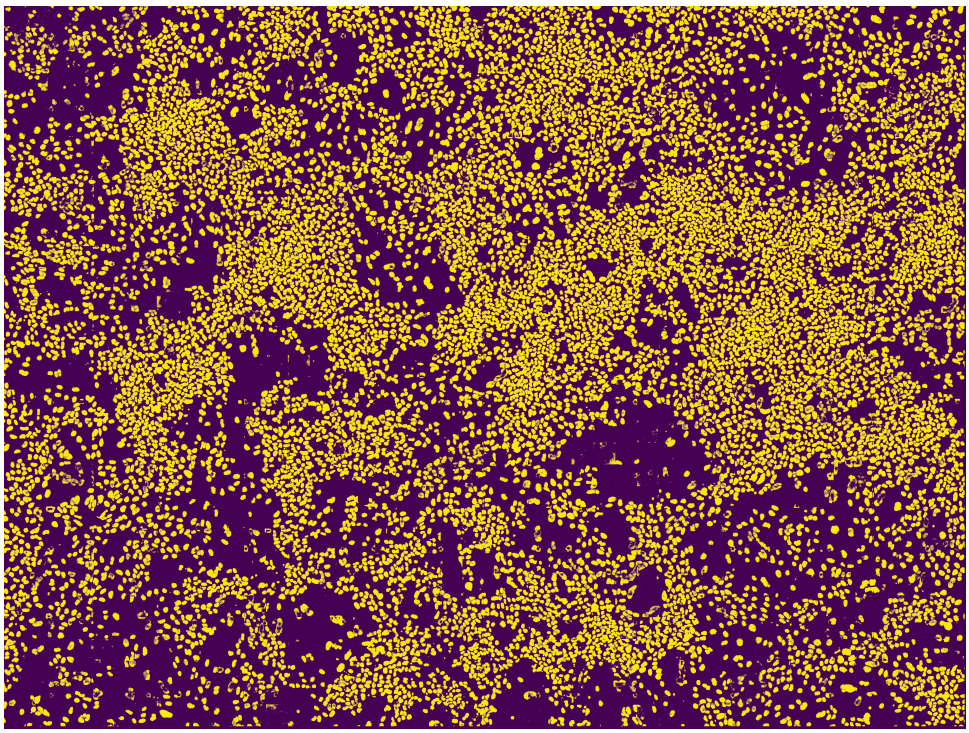


Figure 7: Segmentation of the big picture obtained by merging the $128 \times 128$ unpadded tiles

## 6.2 Histological images of cancerous tissues

As a second application, we downloaded the dataset of the Kaggle challenge [5]. This dataset contained histological images of various cancerous tissues, from kidneys, to livers, to lungs and more. We show below the segmented images obtained through the network on this dataset.

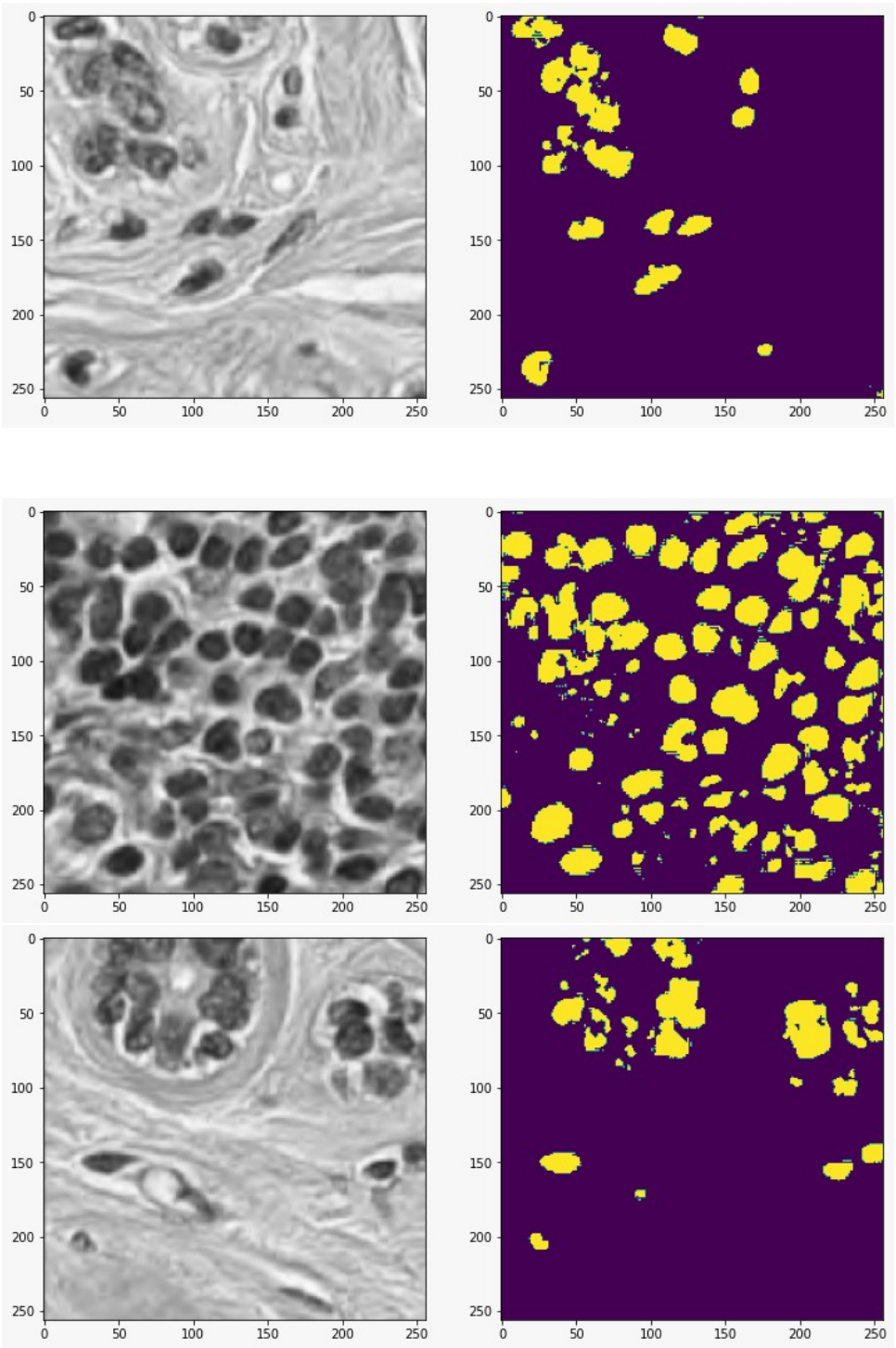



Figure 8: Performance of our network on some instances of the dataset [5], very different from the training one.

We could not quantify the reliability of the segmentation through the f1 score, due to the absence of a reliable *ground truth* of the images. Let us stress that the images used in this application are way different form the ones used during the training session; despite the little accuracy we therefore considered the solution to the segmentation problem acceptable at glance, meaning that the network has abstracted the problem's global features.

# 7 Conclusions

Let us retrace the steps made during the realization of this project.
The aim was to develop a deep neural network able to produce reliable segmentation masks of biomedical images. Starting as a basis from the article [4] we have manipulated both the Kaggle dataset [1] and the neural architecture in order to obtain $256 \times 256$ RGB-images as input and $256 \times 256$ black and white masks as output. The bigger ones have been cropped to create new data images and the smaller symmetrically padded.
One of the crucial points of our project resides in the data augmentation made in the dataset class. We have implemented five random transformations that leave the learning process untouched but yield to an infinite number of data and as a consequence the near impossibility of achieving overfitting.
The satisfactory 0.93 F1 score reached after ca. 40 epochs of training with the optimal model comes from the solid structure of the net; the pretrained batchnormed vgg16_bn encoder decomposes and underlines the features necessary to accomplish the segmentation task; while the concatenated decoder grasps the little details lost in the first portion of the model.
The collaboration between PyTorch an Optuna allowed us to handle hyperparameter tuning; the results have shown best performances using BCE with logistic loss, coupled with the stochastic gradient descent, as optimizer. We set 0.9 momentum and a warm restart scheduler for the learning rate.
As a personal observation, we would like to note the network's ability to underline the segmented blobs' edges, a task that not always the human being manages to accomplish. Maybe this could also explain the reason why we cannot reach a higher F1-score during our experiments, in fact, we noticed that sometimes the target masks were less accurate than the output of our model.

# References


[1] https://www.kaggle.com/c/data-science-bowl-2018.
Kaggle, 2018 Data Science Bowl: Find the nuclei in divergent images to advance medical discovery

[2] Igor Rafael S. Valente, Edson Cavalcanti Neto, Victor Hugo C. Albuquerque.
*Automatic 3D pulmonary nodule detection in CT images: A survey.*
Computer Methods and Programs in Biomedicine, February 2016.

[3] Bin Yang, Junjie Yan et al.
*Convolutional Channel Features.*
ICCV paper, 2015.

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

[5] https://www.kaggle.com/andrewmvd/cancer-inst-segmentation-and-classification.
Kaggle, 2020: Cancer Instance Segmentation and Classification 1