Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

Dataset Generation for the Training of Neural Networks Oriented toward Histological Image Segmentation

Supervisor:
Dr. Enrico Giampieri

Co-supervisor:
Dr. Nico Curti

Submitted by:
Alessandro d'Agostino

Academic Year 2019/2020

13

Abstract

~~Abstract. ...~~The project is inspired by an actual problem of timing end accessibility in the analysis of histological samples in the health-care system. In this project, I face the problem of synthetic histological image generation for the purpose of training Neural Networks for the segmentation of real histological images. The method I propose is based on the replication of the traditional specimen preparation technique in a virtual environment. The first step is the creation of a 3D virtual model of a region of the target human tissue. The model should encapture all the key features of the tissue, and the richer it is the better will be the yielded result. The second step is to perform a sampling of the model through a virtual sectioning process, which produces a first image of the section, which will act as a segmentation mask. This image is then processed with different tools to achieve a histological-like aspect. The most significant contribution is given by the action of a style transfer neural network that implants the typical visual texture of a histological sample onto the synthetic image. This procedure is presented in detail for two specific models of human tissue: one of pancreatic tissue and one of dermal tissue. The two resulting images compose a pair of images and corresponding ground-truth image, which is perfectly suitable for a supervised learning technique. The generation process is completely automatized and does not require the intervention of any human operator, hence it can be used to produce arbitrary large datasets. The synthetic images are inevitably less complex than the real samples and they offer an easier segmentation task to solve for the NN. However, the synthetic images are very abundant, and the training of a NN can take advantage of this feature, following the so-called curriculum learning strategy.

Table of Contents

71

Introduction

In the last decades, the development of Machine Learning (ML) and Deep Learning (DL) techniques has contaminated every aspect of the scientific world, with interesting results in many different research fields. The biomedical field is no exception to this and a lot of promising applications are taking form, especially as Computer-Aided Detection (CAD) systems which are tools for the support ~~of~~ for physicians during the diagnostic process. Medical doctors and the healthcare system in general collect a huge amount of data from patients during all the treatment, screening, and analysis activities in many different shapes, from anagraphical data to blood analysis to clinical images.

In fact in medicine, the study of images is ubiquitous and countless diagnostic procedures rely on it, such as X-ray imaging (CAT), nuclear imaging (SPECT, PET), Magnetic resonance, and visual inspection of histological specimens after biopsies. The branch of artificial intelligence in the biomedical field that handles image analysis to assist physicians in their clinical decisions goes under the name of Digital Pathology Image Analysis (DPIA). In this thesis work, I want to focus on some of the beneficial aspects introduced by DPIA in the histological images analysis and some particular issues in the development of DL models able to handle this kind of procedure.

Nowadays the great majority of analysis of histological specimens occurs through visual inspection, carried out by highly qualified experts. Some analysis, as cancer detection, requires the ability to distinguish if a region of tissue is healthy or not with high precision in very wide specimens. This kind of procedure is typically very complex and requires prolonged times of analysis besides substantial economic efforts. Furthermore, the designated personnel for this type of analysis is often limited, leading to delicate issues of priority assignment while scheduling analysis, based on the estimated patient's clinical development. Some sort of support to this analysis procedure is therefore necessary.

The problem of recognizing regions with different features within an image and detect their borders is known in computer vision as the segmentation task, and it's quite widespread with countless different applications, allowing a sort of automatic image interpretation. In ML the segmentation problem is usually faced as a supervised task, hence the algorithm in order to be trained properly requires an appropriate quantity of pre-labeled images, from which learn the rules through which distinguish different regions. This means that the development of segmentation algorithms for a specific ap-

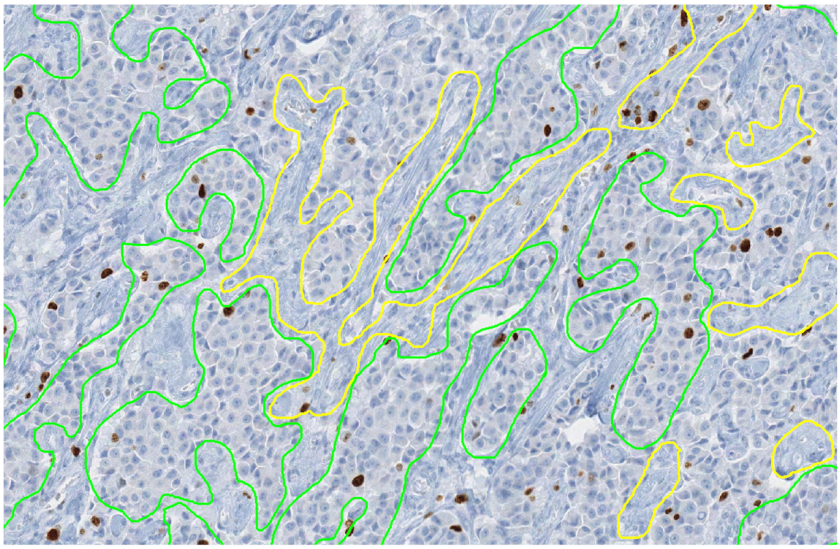


Figure 1: *Interleaving of tumor (green annotation) and non-tumor (yellow annotation) regions [28].*

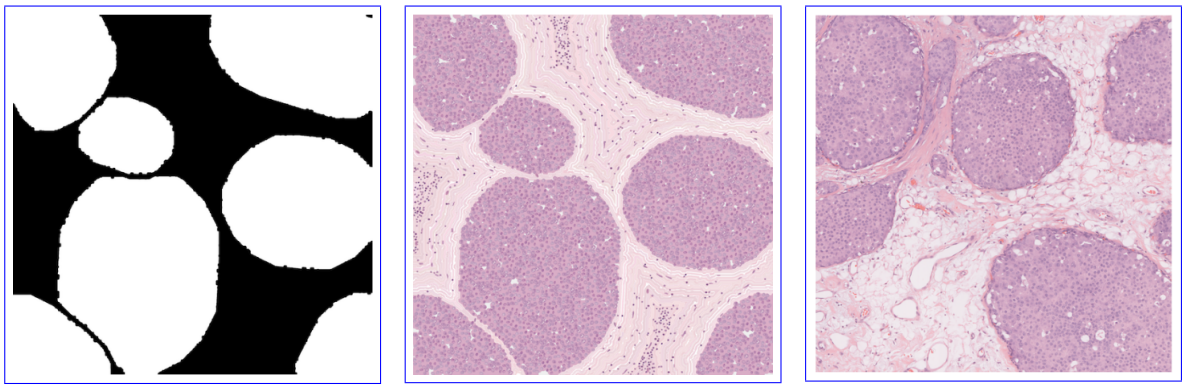
plication, as would be the one on histological images, would require a lot of starting material, previously analyzed from the same qualified expert encharged of the visual inspection mentioned before. A human operator thus is required to manually track the boundaries, for example, between healthy and tumoral regions within a sample of tissue and to label them with their identity, as in Figure 1. The more the algorithm to train is complex the more starting material is required to adjust the model's parameters and reach the desired efficacy.

The latest developed segmentation algorithms are based on DL techniques, hence based on the implementation of intricated Neural Networks (NN) which process the input images and produce the corresponding segmentation. Those models are typically very complex, with millions of parameters to adjust and tune, therefore they need a huge amount of pre-labeled images to learn their segmentation rules. This need for data is exactly the main focus of my thesis work. The shortage of ground truth images is indeed one of the toughest hurdles to overcome during the development of DL-based algorithms. Another important aspect to bear in mind is the quality of the ground truth material. It's impossible for humans to label boundaries of different regions with pixel-perfect precision, while for machines the more precise is the input the more effective is the resulting algorithm.

Different approaches have already been explored to overcome this problem, and they are mainly based on the generation of synthetic labeled data to use during the training phase. Some techniques achieve data augmentation manipulating already available images and then generating *new* images, but as we will see ~~later~~ this approach suffers from

different issues. ~~The technique that~~ Here, I want to make an overview of some other interesting works on the generation of synthetic histological images, which have followed completely different paths and strategies from mine.

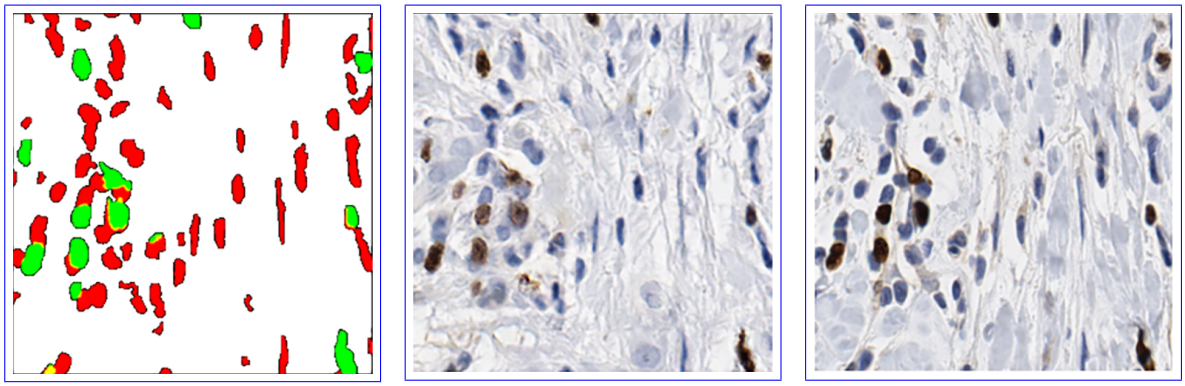
The first work I want to cite is a work from Ben Cheikh *et al.* from 2017 [9]. In this work, they present a methodology for the generation of synthetic images of different types of breast carcinomas. They propose a method completely based on two-dimensional morphology operation, as successive image dilations and erosions. With the modulation of a very restricted number of parameters, regulating the abundance of the objects, their distribution in the image, and their shapes they are able to reconstruct realistic images reflecting different histological situations. In Figure 2 is shown an example of generated material besides a real histological H&E stained sample. Starting from a generated segmentation mask which defines the *tumoral pattern* the production of the synthetic image passes through successive steps, as the generation of characteristic collagen fibers around the structure, the injection of all the immune system cells, and some general final refinements.



(a) *(b)* *(c)*

Figure 2: *Example of generated tumoral pattern (left), which acts as segmentation mask, of generated image (center) and a real example of the tissue to recreate, from [9].*

The second work I want to mention is based on a DL-base technique, which approaches synthetic image generation using a specific cGAN architecture inspired to the "U-net" [38] model, as will be described in Figure 1.13 in section 1.3.1. This model works with Ki67 stained samples of breast cancer tissue, and it is able to generate high-fidelity images starting from a given segmentation mask. Those starting segmentation masks tough are obtained through the processing of other real histological samples, via a nuclei-detection algorithm. The differences between real and synthetic samples are imperceptible, and the material generated in this work has effectively fooled experts, who qualified it has indistinguishable from the real one. In Figure 3 an example of a real image, a generated



(a) *(b)* *(c)*

Figure 3: *Example of generated tumoral pattern (left), which acts as segmentation mask, of generated image (center) and a real example of the tissue to recreate, from [38].*

one, and their corresponding segmentation mask.

Both of the two before-mentioned strategies produces realistic (or even perfect) results, but they are based onto considerations and analysis limited only to the aspect of the images. In the first work, the segmentation mask is produced in an almost full-random way, while in the second the segmentation mask is extracted starting from an actual real histological samples. The target of the present work instead lies in between those two approaches, and wants to produce randomized new images following a plausible modelization based on physical and histological considerations.

The technique I propose in this work follows a generation from scratch of entire datasets suitable for the training of new algorithms, based on the 3D modelization of a region of human tissue at the cellular level. The entire traditional sectioning process, which is made on real histological samples, is recreated virtually on this virtual model. This yields pairs of synthetic images with their corresponding ground-truth. Using this technique one would be able to collect sufficient material for the training (the entire phase or the preliminary part) of a model, overcoming the shortage of hand-labeled data.

The 3D modeling of a region of particular human tissue is a very complex task, and it is almost impossible to capture all the physiological richness of a histological system. The models I implemented thus are inevitably less sophisticated respect the target biological structures. I'll show two models: one of pancreatic tissue and another of dermal tissue, besides all the tools I used and the choices I made during the designing phase.

In order to present organically all the steps of my work the thesis is organized in chapters as follows:

~~Structure~~ [1. Theoretical Background].

In this chapter, I will describe how real histological images are obtained and their

digitalization process works. Afterward, I will introduce the reader to the Deep Learning framework, explaining the key elements of this discipline and how they work. Finally, I will dedicate a section to the image segmentation problem, and the state of the art of segmentation DL-based algorithms, with particular attention to the applications in the bio-medical field.

~~Of the~~ [2. Technical Tools for Model Development].
I will dedicate this chapter to the thorough description of every technical tool I needed during the designing phase of this project. The development has required the harmonization of many different technologies and mathematical tools, some of which not so popular. In this chapter, I will also describe my working environment.

~~Thesis~~

[3. Tissue Models Development].
This third chapter is the heart of the project. I will describe in detail all the steps necessary to create the two models, one of pancreatic tissue and the other of dermal tissue, and how I am able to produce the synthetic images. The first section is devoted to the modeling of the histological structures, while the second in entirely dedicated to the sectioning process and the subsequent refinements to the images.

193

Chapter 1

~~Histological Images~~ ~~Analysis~~Theoretical Background

In this first chapter, I will depict the theoretical context of the work. Section 1.1 will be dedicated to histological images, and the different techniques used to prepare the samples to analyze. Histological images recover a fundamental role in medicine and are the pillar of many diagnosis techniques. This discipline borns traditionally from the optical inspection of the tissue slides using a microscope, and it is gradually developing and improving with the advent of computers and digital image processing. It is important tough to understand how the samples are physically prepared, the final target of this work is in fact the virtual reconstruction of this process. In section 1.2 I will introduce the Deep Learning framework and describe how a Neural Network works and actually learns. The most advanced techniques for the automatic image processing implement Deep Learning algorithm, and understanding the general rules behind this discipline is crucial for a good comprehension of this work. In section 1.3 I will discuss in particular the problem of image segmentation and how it is tackled with different Neural Network architectures, showing what it is the state of the art of this research field.

1.1 Histological Images

Modern histopathology is essentially based on the careful interpretation of microscopic images, with the intention of correctly diagnose patients and to guide therapeutic decisions. In the last years, thanks to the quick development of scanning techniques and image processing, the discipline of histology have seen radical improvements: the main of which undoubtedly is the passage from the microscope's oculars to the computer's screen. This digitalization process has brought several advantages, that were previously impossible in classical histology, like telepathology and remote assistance in diagnosis processes, the integration with other digitalized clinical workflows, and patients' history,

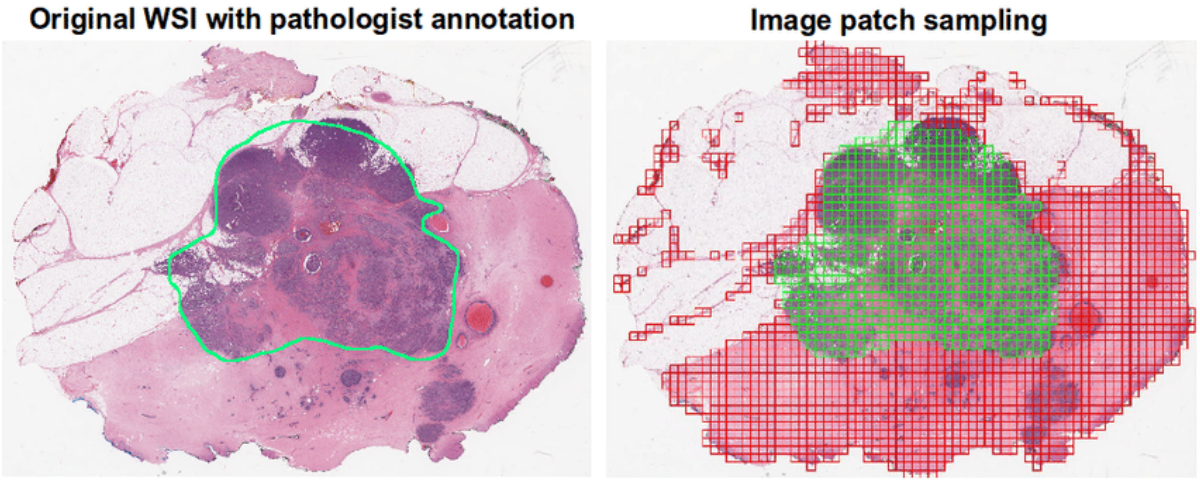


Figure 1.1: *An example of whole slide image, with its grid decomposition in patches. It is visible the correspondence between a region of intereste maually anotated and the patches that matches that region. From [13]*

and most importantly the opening to applications of artificial intelligence. The name Whole Slide Imaging (WSI) refers to the modern virtual microscopy discipline, which consists of scanning a complete microscope slide and creating a single high-resolution digital file. This is commonly achieved by capturing many small high-resolution image tiles or strips and then montaging them to create a full image of a histological section. The four key steps of this process are image acquisition (scansion), editing, and on-screen image visualization.

In the field of Digital Pathology (DP) an essential concept in image understanding is the magnification factor, which indicates the scale of representation of the image and allows dimension referencing. This factor is usually indicated as the magnification power of the microscope's lenses used during the analysis. After the digitalization process, this original magnification factor is prone to change, depending on the resolution of the visualization screen. Therefore, image resolution is measured in μm per pixel, and it is set by the different composition of the acquisition chain, as the optical sensor and the lenses. Histological scanner are usually equipped with $20\times$ or $40\times$ objectives, which correspond to 0.5 and 0.25 mm/pixels resolution values. Lenses with $20\times$ magnification factor are the most suitable for the great majority of histopathological evaluations, and it is the golden standard for scansions, for its good trade-off between image quality and time of acquisition. Scansions with $40\times$ magnification could increase four-fold acquisition and processing time, final file's dimension, and storage cost. A single WSI image, acquired with $20\times$ will occupy more than 600 MB alone.

Despite the WSI is a relatively mature discipline, it still struggles to integrate itself in the standard primitive diagnosis phase in histopathological laboratories. This is primar-

ily due to some disadvantages, like images'resolution, image compression's artifacts, and auto-focusing algorithms, which plays a key role in the specimen interpretation. Furthermore, the scansion of histological samples is an additional step in the analysis which takes time. Despite the technological improvements the average time for the acquisition of a sample is around 5/10 minutes, depending on the number of slices in the slide, for just a single level of magnification. While in traditional histology, the pathologist has access to all the magnification levels at the same time. The real advantage, in fact, is in the long term. Once the images have been acquired they can be archived and consulted remotely almost instantaneously, helping clinical analysis and allowing remote assistance (telemedicine). Furthermore, the images now can be processed by artificial intelligence algorithms, allowing the application of technologies like Deep Learning which could revolutionize the research field, as already has been on many different disciplines in the scientific world.

In order to allow to automatically process, such big images as the ones obtained through WSI, it is necessary to subdivide them in smaller patches. The dimension of which should be big enough to allow interpretation and to preserve a certain degree of representability of the original image. In Figure 1.1 is shown an example of whole slide image, with its grid decomposition in patches. If the patches are too small, it should be over-specified for a particular region of tissue, loosing its general features. This could lead the learning algorithm to misinterpretation. However, this is not an exclusive limit of digital pathology, for a human pathologist would be impossible too to make solid decisions on a too limited sample of tissue. After the subdivision in patches, a typical process for biomedical images is the so-called *data augmentation* of images, that is the process of creating re-newed images from the starting material through simple geometrical transformations, like translation, rotation, reflection, zoom in/out.

The analysis of histological images usually consists in detecting the different components in the samples and to recognise their arrangment as an healthy or pathological pattern. It is necessary to recognize every sign of vitality of the cells, evaluating the state of the nucleus. There are many addittional indicators to consider like the presence of inflammatory cells or tumoral cells. Furthermore, samples taken from different part of the human body present completely different characteristics, and this increase greatly the complexity of the analysis. A reliable esamination of a sample thus require a careful inspection made by an high qualified expert. The automatization of this procedure would be extremily helpful, giving an increadible boost both in timing and in accessibility. However, this is not a simple task and in section 1.3.1 I will show some actual model for biomedical image processing in detail.

1.1.1 Slides Preparation for Optic Microscopic Observation

In modern, as in traditional, histology regardless on the final support of the image the slide has to be physically prepared, starting from the sample of tissue. The sample and

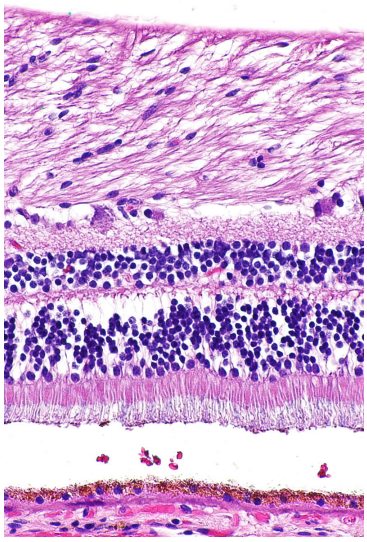


Figure 1.2: *A sample of tissue from a retina (a part of the eye) stained with hematoxylin and eosin, cell nuclei stained blue-purple and extracellular material stained pink.*

slide preparation is a crucial step for histological or cytological observation. It is essential to highlight what needs to be observed and to *immobilize* the sample at a particular point in time and with characteristics close to those of its living state. There are five key steps for the preparation of samples [4]:

1) Fixation is carried out immediately after the removal of the sample to be observed. It is used to immobilize and preserve the sample permanently in as life-like state as possible. It can be performed immersing the biological material in a formalin solution or by freezing, so immersing the sample in a tissue freezing medium which is then cooled in liquid nitrogen.

2) Embedding if the sample has been stabilazed in a fixative solution, this is the subsequent step. It consists in hardening the sample in a paraffin embedding medium, in order to be able to carry out the sectioning. It is necessary to dehydrate the sample beforehand, by replacing the water molecules in the sample with ethanol.

3) Sectioning Sectioning is performed using microtomy or cryotomy. Sectioning is an important step for the preparation of slides as it ensures a proper observation of the sample by microscopy. Paraffin-embedded samples are cut by cross section, using a microtome, into thin slices of 5 μm. Frozen samples are cut using a cryostat. The frozen sections are then placed on a glass slide for storage at -80°C. The choice of these preparation conditions is crucial in order to minimize the artifacts. Paraffin embedding is favored for preserving tissues; freezing is more suitable for preserving

DNA and RNA and for the labeling of water-soluble elements or of those sensitive to the fixation medium.

4) Staining Staining increases contrasts in order to recognize and differentiate the different components of the biological material. The sample is first deparaffinized and rehydrated so that polar dyes can impregnate the tissues. The different dyes can thus interact with the components to be stained according to their affinities. Once staining is completed, the slide is rinsed and dehydrated for the mounting step.

Hematoxylin and eosin stain (H&E) is one of the principal tissue stains used in histology [44], and it is the most widely used stain in medical diagnosis and is often the gold standard [35]. H&E is the combination of two histological stains: hematoxylin and eosin. The hematoxylin stains cell nuclei blue, and eosin stains the extracellular matrix and cytoplasm pink, with other structures taking on different shades, hues, and combinations of these colors. An example of H&E stained is shown in Figure 1.2, in which we can see the typical colour palette of an histological specimen.

1.2 Introduction to Deep Learning

Deep Learning is part of the broader framework of Machine Learning and Artificial Intelligence. Indeed all the problems typically faced using ML can also be addressed with DL techniques, for instance, regression, classification, clustering, and segmentation problems. We can think of DL as a universal methodology for iterative function approximation with a great level of complexity. In the last decades, this technology has seen a frenetic diffusion and an incredible development, thanks to the always increasing available computational power, and it has become a staple tool in all sorts of scientific applications.

1.2.1 Perceptrons and Multilayer Feedforward Architecture

Like other artificial learning techniques, DL models aim to *learn* a relationship between some sort of input and a specific kind of output. In other words, approximating numerically the function that processes the input data and produces the desired response. For example, one could be interested in clustering data in a multidimensional features space, or the detection of objects in a picture, or text manipulation/generation. The function is approximated employing a greatly complex network of simple linear and non-linear mathematical operations arranged in a so-called Neural Network (typically with millions of parameters). The seed idea behind this discipline is to recreate the functioning of actual neurons in the human brain: their entangled connection system and their "ON/OFF" behavior [41].

The fondamental unit of a neural network is called perceptron, and it acts as a digital counterpart of a human neuron. As shown in Figure 1.3 a perceptron collects in input a series on n numerical signals $\vec{x} = 1, x_1, ..., x_n$ and computes a linear wieghted combination with the weights vectors $\vec{w} = w_0, w_1, ..., w_n$, where w_0 is a bias factor:

$$f(\vec{x}, \vec{w}) = \chi(\vec{x} \cdot \vec{w}). \tag{1.1}$$

The results of this linear combination are given as input to a non-linear function $\chi(x)$ called the activation function. Typical choices as activation function are any sigmoidal function like $sign(x)$ and $tanh(x)$, but in more advanced applications other functions like ReLU [1] are used. The resulting function $f(\vec{x}, \vec{w})$ has then a simple non linear behaviour. It produces a binary output: 1 if the weighted combination is high enough and 0 if it is low enough, with a smooth modulation in-between the two values.

The most common architecture for a NN is the so-called *feed-forward* architecture, where many individual perceptrons are arranged in chained layers, which take as input the output of previous layers along with a straight information flux. More complex architectures could implements also recursive connection, linking a layer to itself, but it should be regarded as sophistication to the standard case. There are endless possibilities

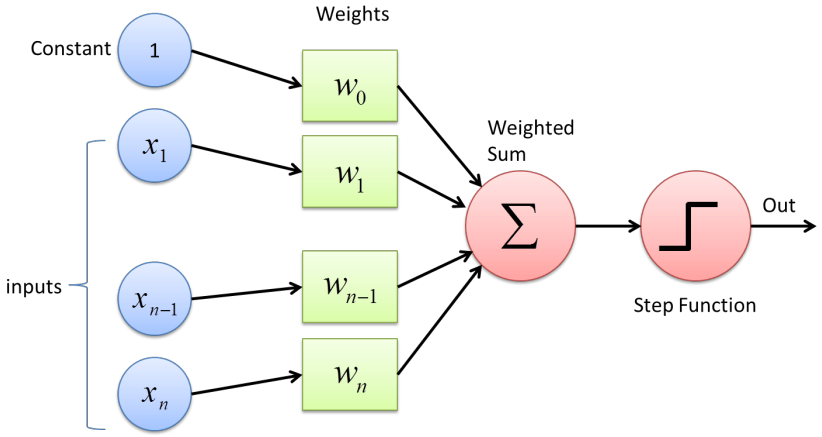


Figure 1.3: *Schematic picture of a single layer perceptron. The input vector is linearly combined with the bias factor and sent to an activation function to produce the numerical "binary" output.*

of combination and arrangement of neurons inside a NN's layer, but the most simple ones are known as *fully-connected* layers, where every neuron is linked with each other neuron of the following layer, as shown in Figure 1.4. Each connection has its weight, which contributes to modulate the overall combination of signals. The training of a NN consists then in the adjustment and fine-tuning of all the network's weights and parameters through iterative techniques until the desired precision in the output generation is reached.

Although a fully connected network represents the simplest linking choice, the insertion of each weight increases the number of overall parameters, and so the complexity of the model. Thus we want to create links between neurons smartly, rejecting the less useful ones. Depending on the type of data under analysis there are many different established typologies of layers. For example, in the image processing field, the most common choice is the convolutional layer, which implements a sort of discrete convolution on the input data, as shown in Figure 1.5. While processing images, the convolution operation confers to the perception of correlation between adjacent pixels of an image and their color channels, allowing a sort of spatial awareness. Furthermore, the majority of traditional computer vision techniques are based on the discrete convolution of images, and on the features extracted from them.

As a matter of principle a NN with just two successive layers, which is called a *shallow* network, and with an arbitrary number of neurons per layer, can approximate arbitrary well any kind of smooth enough function [32]. However, direct experience suggests that networks with multiple layers, called *deep* networks, can reach equivalent results exploiting a lower number of parameters overall. This is the reason why this discipline goes under the name of *deep* learning: it focuses on deep networks with up

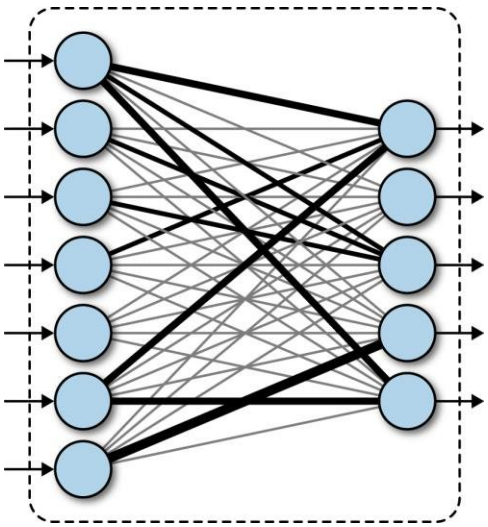


Figure 1.4: *Schematic representation of a fully connected (or dense) layer. Every neuron from the first layer is connected with every output neuron. The link thickness represent the absolute value of the combination weight for that particular value.*

to tents of hidden layers. Such deep structures allow the computation of what is called deep features, so features of the features of the input data, that allows the network to easily manage concepts that would be bearly understandable for humans.

1.2.2 Training of a NN - Error Back-Propagation

Depending on the task the NN is designed for, it will have a different architecture and number of parameters. Those parameters are initialized to completely random values, tough. The training process is exactly the process of seeking iteratively the right values to assign to each parameter in the network in order to accomplish the task. The best start to understanding the training procedure is to look at how a supervised problem is solved. In supervised problems, we start with a series of examples of true connections between inputs and correspondent outputs and we try to generalize the rule behind those examples. After the rule has been picked up the final aim is to exploit it and to apply it to unknown data, so the new problem could be solved. In opposition to the concept of supervised problems, there are the *unsupervised* problems, where the algorithm does not try to learn a rule from a practical example but try to devise it from scratch. A task typically posed as unsupervised is clustering, when different data are separated in groups based on the values of their features in the feature space. Usually, only the number of groups is taken in input from the algorithm, and the subdivision is completely performed by the machine. In the real world, by the way, there are many different and creative

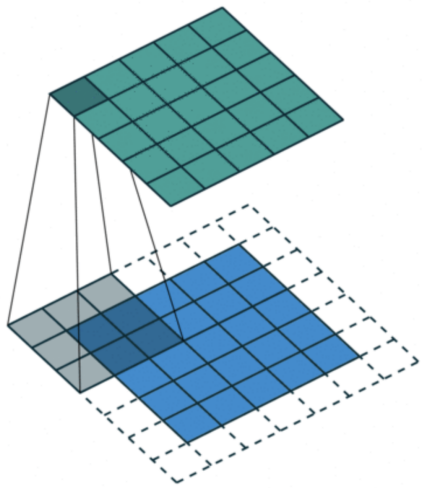


Figure 1.5: *Schematical representation of a convolutional layer. The input data are processed by a window kernel that slides all over the image. This operation can recreate almost all the traditional computer vision techniques, and can overcome them, creating new operations, which would be unthinkable to hand-engineered.*

shapes between pure supervised and pure unsupervised learning, based on the actual availability of data and specific limitations to the individual task.

An interesting mention in this regards should be made about *semi-supervised* learning, which is typically used in bio-medical applications. This learning technique combines a large quantity of unlabeled data during training with a limited number of pre-labeled example. The blend between data can produce considerable improvement in learning accuracy, and leading to better results respect to pure unsupervised techniques. The typical situation of usage of this technique is when the acquisition of labeled data requires a highly trained human agent (as an anatomo-pathologist) or a complex physical experiment. The actual cost of building entire and suitable fully labeled training sets in these situations would be unbearable, and semi-supervised learnign comes in great practical help.

Another important training technique which worth mentioning is the so-called *curriculum learning* [5]. This is a learning technique inspired by the typical learning curve human being and animal, that are used to face problems of always increasing difficulty while learning something new or a new skill. The example presented to the Neural Network are not randomly presented but organized in a meaningful order which illustrates gradually more concepts, and gradually more complex ones. The experiments show that through curriculum learning significant improvements in generalization can be achieved. This approach has both an effect on the speed of convergence of the training process to a

Figure 1.6: *Sample grid of images from the CIFAR10 dataset. Each one of the 32 × 32 image is labeled with one of the ten classes of objects: plane, car, bird, cat, deer, dog, frog, horse, ship, truck.*

minimum and, in the case of non-convex criteria, on the quality of the local minima obtained. This technique is of particular interest for this work: the generated images, which will be shown in section 3.3 , will offer a segmentation task much less complex respect to the analysis of real histological tissue. In the optical of training a DL-based model the abundantly produced images can be used for the preliminary phase of training, setting aside the more complex and more valuable hand-labeled images for the finalization of the training process.

A good example of supervised problems tough is the classification of images. Let's assume we have a whole dataset of pictures of different objects (as cats, dogs, cars, etc.) like the CIFAR10 [22] dataset. This famous dataset is made of over $60K$ labeled colored images 32×32 divided into 10 categories of objects as shown in Figure 1.6. We could be interested in the creation of a NN able to assign at every image its belonging class. This NN could be arbitrarily complex but it certainly will take as input a 32×32×3 RGB image and the output will be the predicted class. A typical output for this problem would be a probability distribution over all the 10 classes like:

$$\vec{p} = (p_1, p_2, \ldots, p_{10}), \tag{1.2}$$

$$\sum_{i=1}^{10} p_i = 1, \tag{1.3}$$

and it should be compared with the true label, that is represented just as a binary sequence $\vec{t}$ with the bit correspondent to the belonging class set as 1, and all the others value set to 0:

$$\vec{t} = (0, 0, \ldots, 1, \ldots, 0, 0). \tag{1.4}$$

Every time an image is given to the model an estimate of the output is produced. Thus, we need to measure the *distance* between that prediction and the true value, to quantify the error made by the algorithm and try to improve the model's predictive power. The functions used for this purpose are called loss functions. The most common choice is the Mean Squared Error (MSE) function that is simply the averaged L^2 norm of the difference vector between $\vec{p}$ and $\vec{t}$:

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(t_i - p_i)^2. \tag{1.5}$$

Let's say the NN under training has L consecutive layers, each one with its activation function f^k and its weights vector $\vec{w}^k$, hence the prediction vector $\vec{p}$ could be seen as the result of the concesutive, nested, application through all the layers:

$$\vec{p} = f^L(\vec{w}^L \cdot (f^{L-1}(\vec{w}^{L-1} \cdot ... \cdot f^1(\vec{w}^1 \cdot \vec{x})))). \tag{1.6}$$

From both equations 1.5 and 1.6 it is clear that the loss function could be seen as a function of all the weights vectors of every layer of the network. So if we want to reduce the distance between the NN prediction and the true value we need to modify those weights to minimize the loss function. The most established algorithm to do so for a supervised task in a feed-forward network is the so-called *error back-propagation*.

The back-propagation method is an iterative technique that works essentially computing the gradient of the loss function with respect to the weights using the derivative chain rule and updating by a small amount the value of each parameter to lower the overall loss function at each step. Each weight is *moved* counter-gradient, and summing all the contribution to every parameter the loss function approaches its minimum. In equation 1.7 is represented the variation applied to the j^{th} weight in the i^{th} layer in a single step of the method:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}, \tag{1.7}$$

where E is the error function, and η is the *learning coefficient*, that modulate the effect of learning through all the training process. This iterative procedure is applied completely to each image in the training set several times, each time the whole dataset is reprocessed is called an *epoch*. The great majority of the dataset is exploited in the training phase

to keep running this trial and error process and just a small portion is left out (typically 10% of the data) for a final performance test.

The loss function shall inevitably be differentiable, and its behavior heavily influences the success of the training. If the loss function presents a gradient landscape rich of local minima the gradient descent process would probably get stuck in one of them. More sophisticated algorithms capable of avoiding this issue have been devised, with the insertion of some degree of randomness in them, as the Stochastic Gradient Descent algorithm, or the wide used *Adam* optimizer [21].

While Error-Back Propagation is the most established standard in DL applications, it suffers from some problems. The most common one is the so-called vanishing or exploding gradient issue, which is due to the iterative chain derivation through all the nested level of composition of the function. Withouth a careful choice of the right activation function and the tuning of the learning hyper-parameters it is very easy to bump into this pitfall. Furthermore the heavy use of derivation rises the inability to handle non-differentiable components and hinders the possibility of parallel computation. However, there are many alternative approaches to network learning beside EBP. The Minimization with Auxiliary Variables (MAV) method builds upon previously proposed methods that break the nested objective into easier-to-solve local subproblems via inserting auxiliary variables corresponding to activations in each layer. Such method avoids gradient chain computation and the potential issues associated with it [10]. A further alternative approach to train the network is the Local Error Signals (LES), which is based on layer-wise loss functions. In [29], is shown that layer-wise training can approach the state-of-the-art on a variety of image datasets. It is used a single-layer sub-networks and two different supervised loss functions to generate local error signals for the hidden layers, and it is shown that the combination of these losses helps with optimization in the context of local learning.

The training phase is the pulsing heart of a DL model development and it could take even weeks on top-level computers for the most complicated networks. In fact, one of the great limits to the complexity of a network during the designing phase is exactly the available computational power. There are many more further technical details necessary for proper training, the adjustment of which can heavily impact the quality of the algorithm. However, after the training phase, we need to test the performance of the NN. This is usually done running the trained algorithm on never seen before inputs (the test dataset) and comparing the prediction with the ground-truth value. A good way to evaluate the quality of the results is to use the same function used as the loss function during the training, but there is no technical restriction to the choice of this quality metric. The average score on the whole test set is then used as a numerical score for the network, and it allows straightforward comparison with other models' performances, trained for the same task. All this training procedure is coherently customized to every different application, depending on which the problem is posed as supervised or not and depending on the more or less complex network's architecture. The leitmotif is always

finding a suitable loss function that quantifies how well the network does what it has been designed to do and trying to minimize it, operating on the parameters that define the network structure.

Figure 1.7: *Example of the resulting segmentation mask of an image of an urban landscape. Every interesting object of the image is detected and a solid color region replaces it in the segmentation mask. Every color corresponds to a different class of objects, for example, persons are highlighted in magenta and scooters in blue. The shape and the boundaries of every region should match as precisely as possible the edges of the objects.*

1.3 Deep Learning-Based Segmentation Algorithms

In digital image processing, image segmentation is the process of recognizing and subdividing an image into different regions of pixels that show similar features, like color, texture, or intensity. Typically, the task of segmentation is to recognize the edges and boundaries of the different objects in the image and assigning a different label to every detected region. The result of the segmentation process is an image with the same dimensions of the starting one made of solid color regions, representing the detected objects. This image is called *segmentation mask*. In Figure 1.7 is shown an example of segmentation of a picture of an urban landscape: different colors are linked to different classes of objects like persons in magenta and scooters in blue. This technology has a significant role in a wide variety of application fields such as scene understanding, medical image analysis, augmented reality, etc.

A relatively easy segmentation problem, and one of the first to be tackled, could be distinguishing an object from the background in a grey-scale image, like in Figure 1.8. The easiest technique to perform segmentation in this kind of problem is based on thresholding. Thresholding is a binarization technique based on the image's grey-level histogram: to every pixel with luminosity above that threshold is assigned the color *white*, and vice versa the color *black*. However, this is a very primitive and fallacious, yet very fast method, and it manages poorly complex images or images with un-uniformity in the background.

A lot of other traditional techniques improve this first segmentation method [11]. Some are based on the object's edges recognition, exploiting the sharp change in luminosity typically in correspondence of the boundary of a shape. Other techniques exploit instead a region-growing technology, according to which some *seed* region markers are

Figure 1.8: *Example of the resulting segmentation mask of an image of a fingeprint obtained trhough a thresholding algorithm. The result is not extremely good, but this techinque is very easy to implement and runs very quickly.*

scattered on the image, and the regions corresponding to the objects in the image are grown to incorporate adjacent pixels with similar properties.

Every development of traditional computer vision or of Machine Learning-based segmentation algorithm suffers from the same, inevitable limitation. For every one of those techniques is the designing phase in with the operator should decide precisely which features to extract from the image, like different directional derivatives in the image plane or image entropy, and how to process them for the rest of the analysis. There is thus an intrinsic limitation in the human comprehension of those quantities and in the possible way to combine them. The choice is made on the previous experimental results in other image processing works, and on their theoretical interpretation. Neural Networks instead are relieved from this limitation, allowing themselves to learn which features are best suited for the task and how they should be processed during the training phase. The complexity is then moved on to the design of the DL model and on its learning phase rather than on the hand-engineering design of the features to extract. In Figure 1.9 an example high-level feature extracted from a DL model trained for the segmentation of nuclei in a histological sample. The model learns the typical pattern of arrangement of nuclei, which would have been impossible to describe equally in advance.

1.3.1 State of the Art on Deep Learning Segmentation

Similarly to many other traditional tasks, also for segmentation, there has been a thriving development lead by the diffusion of deep learning, that boosted the performances resulting in what many regards as a paradigm shift in the field [27].

In further detail, image segmentation can be formulated as a classification problem of pixels with semantic labels (semantic segmentation) or partitioning of individual objects (instance segmentation). Semantic segmentation performs pixel-level labeling with a set of object categories (e.g. boat, car, person, tree) for all the pixels in the image, hence it is typically a harder task than image classification, which requires just a single label for

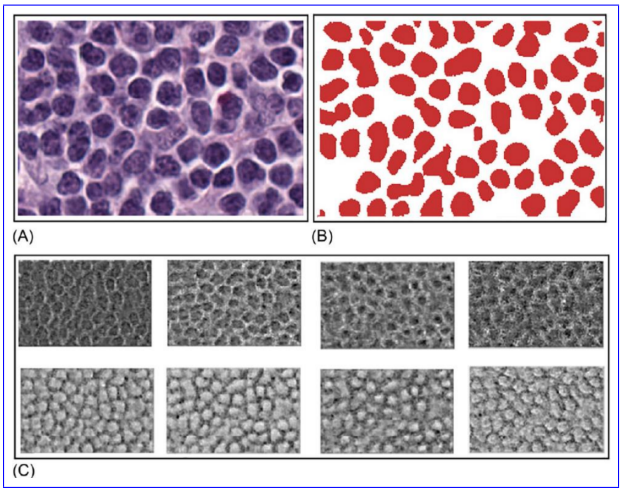


Figure 1.9: *(A) An extract from an histological samples, used as input image for the model. (B) The exact segmentation mask. (C) Example of accentuated features during the training: (1-4) for back-ground recognition, (5-8) for nuclei detection. From [2].*

the whole image. Instance segmentation extends semantic segmentation scope further by detecting and delineating each object of interest in the image (e.g. partitioning of individual nuclei in a histological image).

There are many prominent Neural Network architectures used in the computer vision community nowadays, based on very different ideas such as convolution, recursion, dimensionality reduction, and image generation. This section will provide an overview of the state of the art of this technology and will dwell briefly on the details behind some of those innovative architectures.

Recurrent Neural Networks (RNNs) and the LSTM

The typical application for RNN is processing sequential data, as written text, speech or video clips, or any other kind of time-series signal. In this kind of data, there is a strong dependency between values at a given time/position and values previously processed. Those models try to implement the concept of *memory* weaving connections, outside the main information flow of the network, with the previous NN's input. At each time-stamp, the model collects the input from the current time X_i and the hidden state from the previous step h_{i-1} and outputs a target value and a new hidden state (Figure 1.10). Typically RNN cannot manage easily long-term dependencies in long sequences of signals. There is no theoretical limitation in this direction, but often it arises vanishing (or exploding) gradient problematics during the training phase. A specific type of RNN has been designed to avoid this situation, the so-called Long Short Term Memory (LSTM) [19]. The

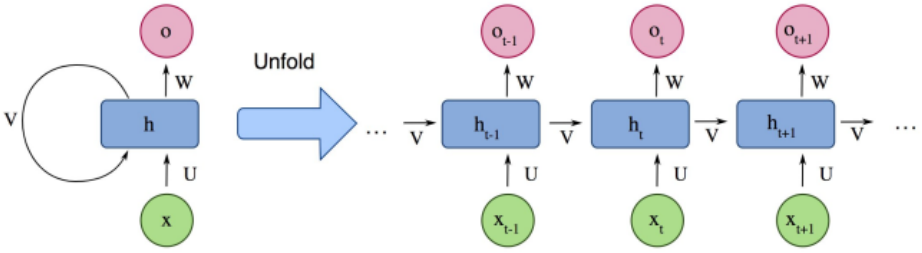


Figure 1.10: *Example of the structure of a simple Recurrent Neural Network from [27].*

LSTM architecture includes three gates (input gate, output gate, forget gate), which regulate the flow of information into and out from a memory cell, which stores values over arbitrary time intervals.

Encoder-Decoder and Auto-Encoder Models

Encoder-Decoder models try to learn the relation between an input and the corresponding output with a two steps process. The first step is the so-called *encoding* process, in which the input x is compressed in what is called the *latent-space* representation $z = f(x)$. The second step is the *decoding* process, where the NN predicts the output starting from the latent-space representation $y = g(z)$. The idea underneath this approach is to capture in the latent-space representation the underlying semantic information of the input that is useful for predicting the output. ED models are widely used in image-to-image problems (where both input and output are images) and for sequential-data processing (like Natural Language Processing, NLP). In Figure 1.11 is shown a schematic representation of this architecture. Usually, these model follow a supervised training, trying to reduce the reconstruction loss between the predicted output and the ground-truth output provided while training. Typical applications for this technology are image-enhancing techniques like de-noising or super-resolution, where the output image is an improved version of the input image. Or image generation problems (e.g. plausible new human faces generation) in which all the properties which define the type of image under analysis should be learned in the representation latent space.

Generative Adversarial Networks (GANs)

The peculiarity of Generative Adversarial Network (GAN) lies in its structure. It is actually made of two distinct and independent modules: a generator and a discriminator, as shown in Figure 1.12. The first module G, responsible for the generation, typically learns to map a prior random distribution of input z to a target distribution y, as similar as possible to the target $G = z \rightarrow y$ (i.e. almost

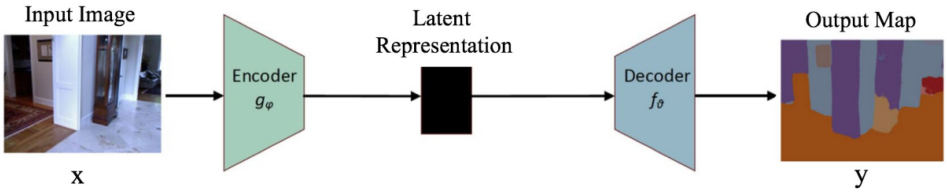


Figure 1.11: *Example of the structure of a simple Encoder-Decoder Neural Network from [27].*

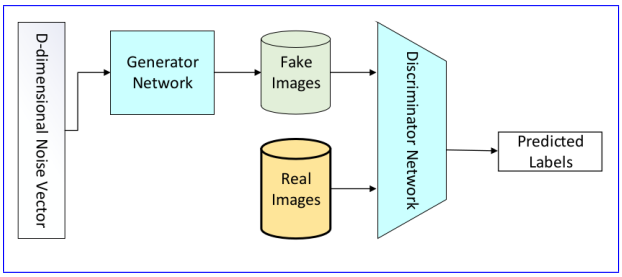


Figure 1.12: *Schematical representation of a Generative Adversarial Networks, form [27].*

any kind of image-to-image problem could be addressed with GANs, as in [20]). The second module, the discriminator D, instead is trained to distinguish between *real* and *fake* images of the target category. These two networks are trained alternately in the same training process. The generator tries to fool the discriminator and vice versa. The name adversarial is actually due to this *competition* within different parts of the network. The formal manner to set up this adversarial training lies in the accurate choice of a suitable loss function, that will look like:

$$L_{GAN} = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_z(z)}[log(1 - D(G(z)))]$$

. The GAN is thus based on a min-max game between G and D. D aims to reduce the classification error in distinguishing fake samples from real ones, and as a consequence maximizing the L_{GAN}. On the other hand, G wants to maximize the D's error, hence minimizing L_{GAN}. The result of the training process is the trained generator G^*, capable of produce an arbitrary number of new data (images, text, or whatever else):

$$G^* = \arg\min_G \max_D L_{GAN}$$

. This peculiar architecture has yielded several interesting results and it has been developed in many different directions, with influences and contaminations with other architectures [20].

Convolutional Neural Networks (CNNs)

As stated before CNNs are a staple choice in image processing DL applications. They mainly consist of three types of layers:

i convolutional layers, where a kernel window of parameters is convolved with the image pixels and produce numerical features maps.

ii nonlinear layers, which apply an activation function on feature maps (usually element-wise). This step allows the network to introduce non-linear behavior and then increasing its modeling capabilities.

iii pooling layers, which replace a small neighborhood of a feature map with some statistical information (mean, max, etc.) about the neighborhood and reduce the spatial resolution.

Given the arrangement of successive layers, each unit receives weighted inputs from a small neighborhood, known as the receptive field, of units in the previous layer. The stack of layers allows the NN to perceive different resolutions: the higher-level layers learn features from increasingly wider receptive fields. The leading computational advantage given by CNN architecture lies in the sharing of kernels' weights within a convolutional layer. The result is a significantly smaller number of parameters than fully-connected neural networks. In section 2.7 will be shown a particular application of this architecture, known as *style-transfer* network, which is a particular algorithm capable of implanting the visual texture of a *style* image onto the content of a different image, producing interesting hybrid images. Some of the most notorious CNN architectures include: AlexNet [23], VGGNet [40], and U-Net [34].

For this work, U-net architecture is particularly interesting. The U-net model was initially developed for biomedical image segmentation, and in its structure reflects characteristics of both CNN and Encoder-Decoder models. Ronneberger et al.[34] proposed this model for segmenting biological microscopy images in 2015. The U-Net architecture is made of two branches, a contracting path to capture context, and a symmetric expanding path (see Figure 1.13). The down-sampling flow is made of a Fully Convolutional Network (FCN)-like architecture that computes features with 3×3 kernel convolutions. On the other hand, the up-sampling branch exploits up-convolution operations (or deconvolution), reducing the number of feature maps while increasing their dimensions. Another characteristic of this architecture is the presence of direct connections between layers of a similar level of compression in compressing and decompressing branches. Those links allow the NN to preserve spatial and pattern information. The Network flow eventually ends with a 1×1 convolution layer responsible for the generation of the segmentation mask of the input image.

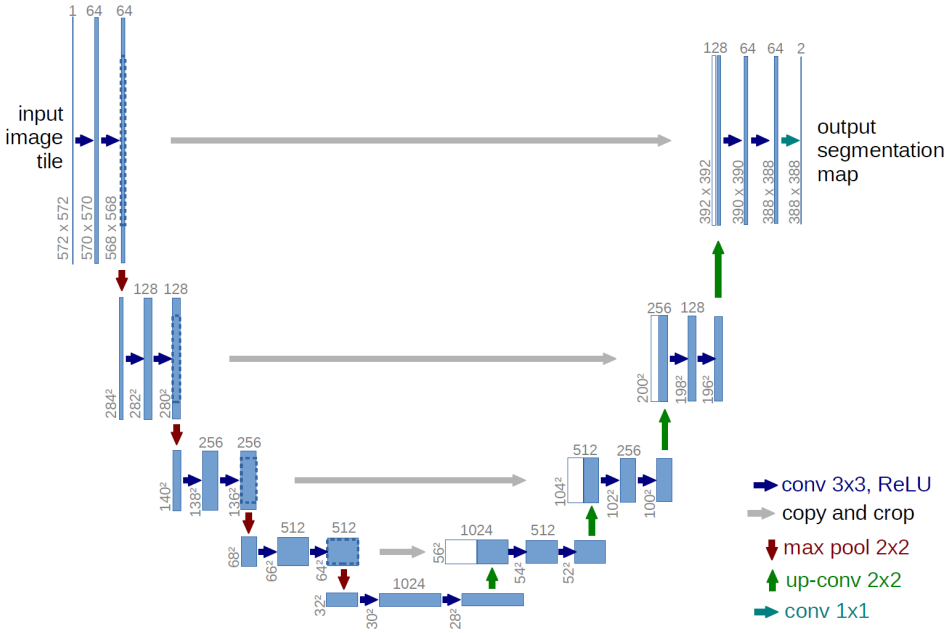


Figure 1.13: *Scheme of the typical architecture of a U-net NN. This particular model was firstly proposed by Ronneberger et al. in [34].*

A recent example of a practical application of a CNN to histological images could be found in [28]. In this work the `Inception v3` is trained on hand-labeled samples of pancreatic tissue (like in Figure 1) to recognise tumoral regions from helthy ones in a pancreatic tissue specimen treated with Ki67 staining. The Inception v3 [33] network is a deep convolutional network developed by Google, trained for object detection and image calssification on the ImageNet dataset [15]. Recognition of tumoral region of Ki67 stained pancreatic tissue samples is based on the detection and counting of some specific marker cells. In Figure 1.14 is shown a pair of original image and the computed segmentation mask, which label in red tumoral regions and in green the healthy ones. This work is based on a technique called `transfer learning`, which consists in the customization and specialization of a pre-trained NN previously trained for similar, but essentially different, tasks. The final part of the training of this version of Inception v3 has been performed of a dataset of 33 whole slide images of Ki67 stained neuroendocrine tumor biopsies acquired from 33 different patients, digitized with a 20× magnification factor and successively divided in 64×64 patches.

1.3.2 Image Segmentation Datasets

Besides the choice of suitable architecture the most important aspect while developing a NN is the dataset on which perform the training process. Let's confine the discussion

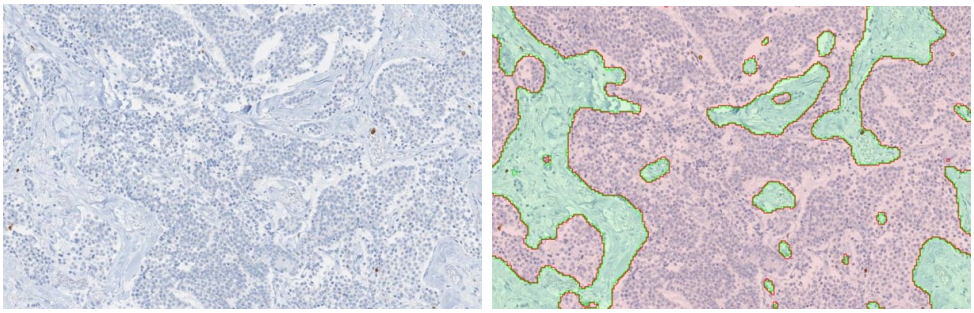


Figure 1.14: *(left) Original image of a Ki67 stained pancreatic tissue sample, (right) the corresponding segmentation mask, which label in red tumoral regions and in green the healthy ones. From [28].*

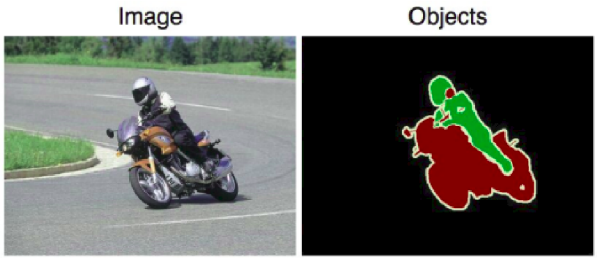


Figure 1.15: *An example image from the PASCAL dataset and its corresponding segmentation mask [17].*

only to image-to-image problems, like segmentation problems. There are a lot of widely used datasets, but I want to mention just a few of them to give the idea of their typical characteristics.

A good example of segmentation is the Cityscapes dataset [12], which is a large-scale database with a focus on semantic understanding of urban street scenes. The dataset is made of video sequences from the point of view of a car in the road traffic, from 50 different cities in the world. The clips are made of 5K frames, labeled with extremely high quality at pixel-level and an additional set of 20K weakly-annotated frames. Each pixel in the segmentation mask contains the semantic classification, among over 30 classes of objects. An example of an image from this dataset is shown in Figure 1.7.

The PASCAL Visual Object Classes (VOC) [17] is another of the most popular datasets in computer vision. This dataset is designed to support the training of algorithms for 5 different tasks: segmentation, classification, detection, person layout, and action recognition. In particular, for segmentation, there are over 20 classes of labeled objects (e.g. planes, bus, car, sofa, TV, dogs, person, etc.). The dataset comes divided into two portions: training and validation, with 1,464 and 1,449 images, respectively. In Figure 1.15 is shown an example of an image and its corresponding segmentation mask.

As last mention I would report t he ImageNet project [15], which is a large visual database designed for use in visual object recognition software research. It consists of more than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. ImageNet contains more than 20,000 categories of objects. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. This kind of competitions is very important for the research field, as it inspire and encourage the development of new models and architectures.

It is worth mentioning that in the medical image processing domain typically the available dataset is definitely not that rich and vast (that is actually the seed of this work) and thus many techniques of data augmentation have been devised, to get the best out of the restricted amount of material. Generally, data augmentation manipulates the starting material applying a set of transformation to create new material, like rotation, reflection, scaling, cropping and shifting, etc. Data augmentation has been proven to improve the efficacy of the training, making the model less prone to overfitting, increasing the generalization power of the model, and helping the convergence to a stable solution during the training process.

Chapter 2

Technical Tools for Model Development

As mentioned in the introduction, this project wants to produce synthetic histological images paired with their corresponding segmentation mask, to train Neural Networks for the automatization of real histological images analysis. The production of artificial images passes through the processing of a three dimensional, virtual model of a histological structure, which is the heart of this thesis work. The detailed description of the development of the two proposed histological models will follow the present chapter and will occupy all the chapter 3. Here I will dwell, instead, on every less common tool employed during the models' designing phase. From the practical point of view, this project is quite articulated and the development has required the harmonization of many different technologies, tools, and code libraries. The current chapter should be seen as a theoretical complement for chapter 3, and its reading is suggested to the reader for any theoretical gap or for any further technical deepening. The reader already familiar with those technical tools should freely jump to the models' description.

All the code necessary for the work has been written in a pure `Python` environment, using several already established libraries and writing by my self the missing code ~~,~~ for some specific applications. I decided to code in `Python` given the thriving variety of available libraries geared toward scientific computation, image processing, data analysis, and last but not least for its ease of use (compared to other programming languages). In each one of the following subsections ~~will be mentioned~~ I will mention the specific code libraries which have been employed in this project for every technical necessity.

2.1 Quaternions

Quaternions are, in mathematics, a number system that expands to four dimensions the complex numbers. They have been described for the first time by the famous mathemati-

cian William Rowan Hamilton in 1843. This number system define three independent *imaginary* units $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ as in (2.1), which allows the general representation of a quaternion $\boldsymbol{q}$ is (2.2) and its inverse $\boldsymbol{q}^{-1}$ (2.3) where a, b, c, d are real numbers:

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1, \tag{2.1}$$

$$\boldsymbol{q} = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}, \tag{2.2}$$

$$\boldsymbol{q}^{-1} = (a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k})^{-1} = \frac{1}{a^2 + b^2 + c^2 + d^2}\,(a - b\boldsymbol{i} - c\boldsymbol{j} - d\boldsymbol{k}). \tag{2.3}$$

Furthermore, the multiplication operation between quaternionn does not benefit from commutativity, hence the product between basis elements will behave as follows:

$$\begin{gathered}\boldsymbol{i}\cdot 1 = 1\cdot\boldsymbol{i} = \boldsymbol{i}, \qquad \boldsymbol{j}\cdot 1 = 1\cdot\boldsymbol{j} = \boldsymbol{j}, \qquad \boldsymbol{k}\cdot 1 = 1\cdot\boldsymbol{k} = \boldsymbol{k} \\ \boldsymbol{i}\cdot\boldsymbol{j} = \boldsymbol{k}, \qquad \boldsymbol{j}\cdot\boldsymbol{i} = -\boldsymbol{k} \\ \boldsymbol{k}\cdot\boldsymbol{i} = \boldsymbol{j}, \qquad \boldsymbol{i}\cdot\boldsymbol{k} = -\boldsymbol{j} \\ \boldsymbol{j}\cdot\boldsymbol{k} = \boldsymbol{i}, \qquad \boldsymbol{k}\cdot\boldsymbol{j} = -\boldsymbol{i}.\end{gathered} \tag{2.4}$$

This number system has plenty of peculiar properties and applications, but for this project, quaternions are important for their ability to represent, in a very convenient way, rotations in three dimensions. The particular subset of quaternions with vanishing real part ($a = 0$) has a useful, yet redundant, correspondence with the group of rotations in tridimensional space SO(3). Every 3D rotation of an object can be represented by a 3D vector $\vec{u}$: the vector's direction indicates the axis of rotation and the vector magnitude $|\vec{u}|$ express the angular extent of rotation. However, the matrix operation which expresses the rotation around an arbitrary vector $\vec{u}$ it is quite complex and does not scale easily for multiple rotations [7], which brings to very heavy and entangled computations.

Using quaternions for expressing rotations in space, instead, it is very convinient. Given the unit rotation vector $\vec{u}$ and the rotation angle θ, the corresponding rotation quaternion $\boldsymbol{q}$ becomes (2.6):

$$\vec{u} = (u_x, u_y, u_z) = u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k}, \tag{2.5}$$

$$\boldsymbol{q} = e^{\frac{\theta}{2}(u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k})} = \cos\frac{\theta}{2} + (u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k})\sin\frac{\theta}{2}, \tag{2.6}$$

$$\boldsymbol{q}^{-1} = \cos\frac{\theta}{2} - (u_x\boldsymbol{i} + u_y\boldsymbol{j} + u_z\boldsymbol{k})\sin\frac{\theta}{2}, \tag{2.7}$$

where in (2.6) we can clearly see a generalization of the Euler's formula for the exponential notation of complex numbers, which hold for quaternions. It can be shown

that the application of the rotation represented by $\boldsymbol{q}$ on an arbitrary 3D vector $\vec{v}$ should be easily expressed as:

$$\vec{v}\,' = \boldsymbol{q}\vec{v}\boldsymbol{q}^{-1}, \tag{2.8}$$

using the Hamilton product defined on quaternions (2.4). This rule raises a very convinient and an extremily scalable way to compute consecutive rotations in space. Given two independent and consecutive rotations represented by the two quaternions $\boldsymbol{q}$ and $\boldsymbol{p}$ applyed on the vector $\vec{v}$ the resulting rotated vector $\vec{v}\,'$ is simply yielded as:

$$\vec{v}\,' = \boldsymbol{p}(\boldsymbol{q}\vec{v}\boldsymbol{q}^{-1})\boldsymbol{p}^{-1} = (\boldsymbol{p}\boldsymbol{q})\vec{v}(\boldsymbol{q}\boldsymbol{p})^{-1}, \tag{2.9}$$

which essentially is the application of the rotation $\boldsymbol{r} = \boldsymbol{q}\boldsymbol{p}$ on the vector $\vec{v}$. This representation is completely coherent with the algebra of 3D rotations, which does not benefit from commutativity in turn.

Given this property, quaternions are indeed widely used in all sorts of applications of digital 3D space design, as for simulations and videogame. The position of an object in the space in simulations is generally given by the application of several independent rotations, typically in the order of a tenth of rotations, which with quaternions is given easily by the product of simple objects. Every other alternative method would imply the use of matrix representation of rotations or other rotation systems as Euler's angles and would eventually make the computation prohibitive.

The use of quaternions in this work will be justified in section 3.1, while speaking of parametric L-systems in 3D space, used to build the backbone of the ramified structure of blood vessels in the reconstruction of a sample of pancreatic tissue.

I was able to find many `Python` libraries for computation with quaternions, but the one I appreciated the most for its interface and ease of use was the `pyquaternion`. With this library, it's immediate the definition of a quaternion by its correspondent rotation vector, and the multiplication between quaternions is straightforward.

2.2 Parametric L-Systems

Lindenmayer systems, or simply L-systems, were conceived as a mathematical theory of plant development [25] in 1968 by Aristid Lindenmayer. Successively, a lot of geometrical interpretations of L-systems were proposed to make them a versatile instrument for modeling the morphology typical of plants and other organic structures. As a biologist, Lindenmayer studied different species of yeast and fungi and worked the growth patterns of various types of bacteria (e.g. as the *cyanobacteria Anabaena catenula*). The main purpose for which L-systems were devised was to allow a formal description of the development of simple multicellular living organisms. Subsequently, the potentiality of these systems was expanded to describe higher-order plants and complex branching structures.

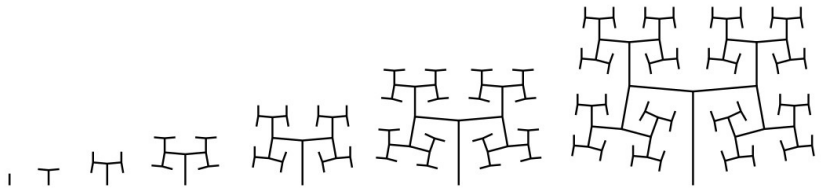


Figure 2.1: *Growth pattern for the space-filling fract-like system, used to mimic the blood vassel bifurcations in sec 3.1.*

An L-system is in general defined by an *axiom* sequence and some development *rules*, which are recursively applied to the sequence and lead its development. The original proposed L-system was fairly simple and shows really well the idea underneath:

$$\begin{aligned} &axiom \ : \ A \\ &rules \ : \ (A \to AB), \quad (B \to A) \end{aligned}$$

where A and B could be any two different patterns in the morphology of an algae, or could be different bifurcations in a ramificated structure. The iterative application of the rules to the axiom sequence, let's say for 7 times, will produce the following sequence:

$$\begin{aligned} n = 0 \ &: A \\ n = 1 \ &: AB \\ n = 2 \ &: ABA \\ n = 3 \ &: ABAAB \\ n = 4 \ &: ABAABABA \\ n = 5 \ &: ABAABABAABAAB \\ n = 6 \ &: ABAABABAABAABABAABABA \\ n = 7 \ &: ABAABABAABAABABAABABAABAABABAABAAB \, . \end{aligned}$$

This kind of tool, as will be shown also in 3.1, is particularly suited for the creation of structures with fractal behavior, and it has been used in this work to create the backbone of the entangled bifurcation in blood vessels in the modelization of pancreatic tissue. In particular, there was the need for a fractal-like space-filling ramification as the one shown in Figures 2.1.

The system in Figure 2.1 represent the successive ramification of a structure which grows adding segments gradually shorter, by a lenght ratio parameter R and inclined of $\delta = \pm 85°$ respect the previous branch. The axiom and the rules that produce this structure are the following:

$$
\begin{aligned}
&axiom \ : \ A \\
&rule_1 \ : \ A \rightarrow F(1)[+A][-A] \\
&rule_2 \ : \ F(s) \rightarrow F(s \cdot R)
\end{aligned}
\tag{2.10}
$$

where A represent the start of a new branch and $F(s)$ represent a branch of lenght s. The presence of a rule which acts dfferently depending on the target object, is an further sophistication respect to the standard L-system. For this reason these systems are called *parametric* L-systems.

The use of standard L-systems turned out to be widespread, and there were a lot of different `Python` libraries at my disposal for coding. By the way, parametric L-systems were not just as popular, and I was not able to find a reliable library on which to build my work. I decided then to code a parametric branching system able to recreate the structure with rules (2.10) at any desired level of iteration. Having created the tool I needed on my own I was able to add all the optional features I would have needed during the development, like an adjustable degree of angular noise in the branch generation.

2.3 Voronoi Tassellation

Voronoi diagrams, or Voronoi decompositions, are space-partitioning systems, which divides an n-dimensional Euclidian space into sub-regions depending on the proximity to a given set of objects. More precisely, given an n-dimensional space and m starting point $p_1, \ldots, p_m$ inside it, the whole space will be subdivided in m adjacent regions. Every point of the space is assigned to the region correspondent to the nearest starting point. In Figure 2.2 is shown a practical example of a Voronoi decomposition of a plane into 20 regions corresponding to the 20 starting points. Informal use of Voronoi diagrams can be traced back to Descartes in 1644, and many other mathematicians after him. But, Voronoi diagrams are named after Georgy Feodosievych Voronoy who defined and studied the general n-dimensional case in 1908 [46].

More precisely, let X be a metric space and d the distance defined on it. Let K be the set of indices and let $(P_k)_{k \in K}$ be the tuple of sites in the space X. The k^{th} Voronoi cell R_k, associated with the site P_k is the set of all the points in X whose distance to P_k is smaller than the distance to any other site P_j, with $j \neq k$, or in other words:

$$
R_k = \{x \in X \mid d(x, P_k) \leq d(x, P_j) \ \forall j \in K, \ j \neq k\}, \tag{2.11}
$$

depending on the notion of distance defined on the space X the final redistribution in subregions will look very differently.

In addition to the choice of the distance function, another fundamental factor is the distribution of sites in the space to be divided. If the points are chosen equally and homogenously distributed the final distribution will appear as a simple regular lattice, while

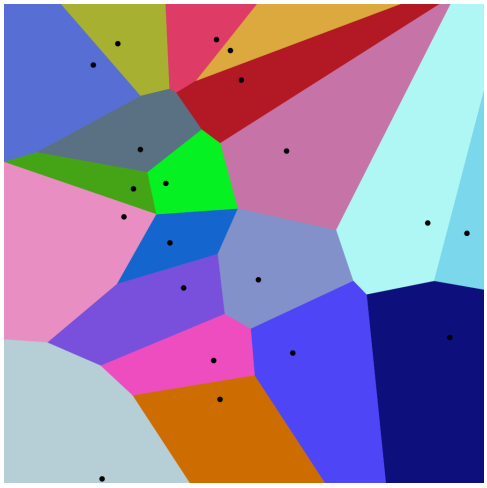


Figure 2.2: *Example of a Voronoi decomposition of a plane into 20 regions corresponding to 20 starting points.*

a completely random distribution of points in the space will provide a decomposition in cells with very different shapes and volumes, as shown in Figure 2.3. Interesting results concerning points from a semi-random distribution will be shown in section 3.1, which leads to decomposition with a good richness in shapes but with the desired homogeneity in volumes.

The Voronoi decomposition has been of great interest in this project for the division of a 3D space in subregions, to recreate the spatial distribution of cells in a sample of human tissue, as will be shown in section 3.1. The formal definition of Voronoi regions (2.11) ensures the convexity of each decomposition's tassel, which in three-dimensional space would be adjacent convex polyhedrons. Every tassel of the decomposition will be represented by a bounded 3-dimensional convex hull [1], with except for those most external cells which are unbounded and requires special attention while using.

The most widespread tool for the computation of Voronoi decompositions in `Python` is contained in the `spatial` submodule of the famous library `SciPy` [45], which is a staple tool for an incredible variety of scientific algorithms. The `Voronoi` object from `Scipy` library offers a very efficient algorithm for space-partitioning, and it has been one of the pillars for the modelization of tissues. Unluckily this module does not easily allow to perform Voronoi decomposition with different definitions of distance functions d other than the Euclidian distance, which would have allowed interesting studies.

[1] See section 2.5 for further details.

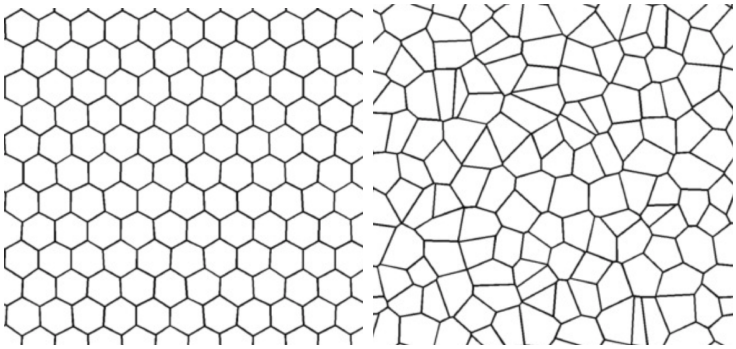


Figure 2.3: *On the right an example of 2D Voronoi decomposition resulting from homogeneously distributed points in the plane. On the left the resulting decomposition obtained from randomly distributed points in the plane, from [3].*

2.4 Saltelli Algorithm - Randon Number Generation

As mentioned in section 2.3, in this project there was the need for quasi-random number generation for the production of Voronoi tessellations. Quasi-random sequences (or low-discrepancy sequences) are patterns of numbers that emulate the behavior of uniform random distributions but have a more homogeneous and quick coverage of the sampling domain, which provides an important advantage in applications as in quasi-Monte Carlo integration techniques, as shown in Figure 2.4. In computer science there is not any possibility of recreating *true* random sequences, hence any stochasticity is completely deterministic in its essence even if produced by very chaotic processes[2]. Indeed, every algorithm for random number generation is completely repeatable given its starting status. Quasi-random sequences are completely deterministic too, but implements more *regular*, well-behaved algorithms.

A first good example to understand the concept of quasi-random generation could be an additive reccurrence, as the following:

$$s_{n+1} = (s_n + \alpha) \bmod 1, \tag{2.12}$$

which for every seed element s_0 and real parameter α produced completely different sequences.

In the bottom line of Figure 2.4 is clearly visible the good and homogeneous coverage of the sampling domain, although it is strongly visible a regular pattern between points, which does not convey an *organic* sensation at all. However, increasing the complexity of

[2]A chaotic process is a deterministic process which has an extremely sensible dependence on its starting conditions. This property mimics very effectively the behavior of true random processes, which are intrinsically forbidden in computer science.

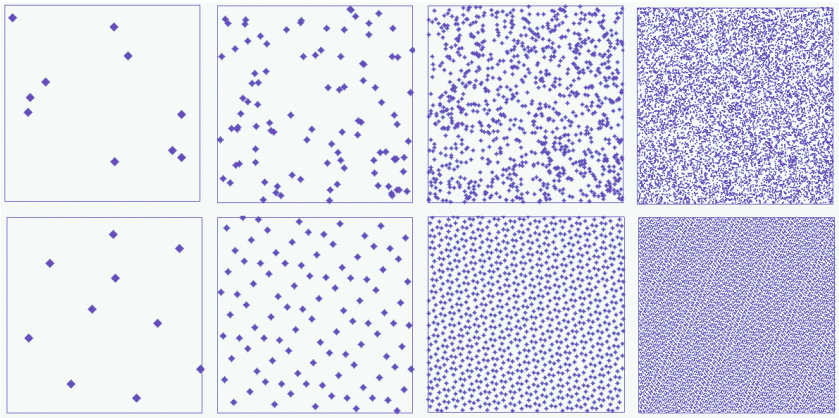


Figure 2.4: *Coverage of the unit square with an additive quasirandom numbers sequence as in 2.12 (up) and for uniformly sampled random numbers (bottom). From left to right: 10, 100, 1000, 10000 points.*

our very simple starting model 2.12 it is possible to overcome this *artificial* appearance of sampled points and to produce very good samples.

A notorious algorithm for quasi-random number generation is the Sobol sequence, introduced by the russian mathematician Ilya M. Sobol in 1967 [43]. In its work, Sobol wanted to construct a sequence x_n of points in the s-dimensional unitary hybercube $I^s = [0,1]^s$ such as for any integrable function f:

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i) = \int_{I_s} f. \tag{2.13}$$

Sobol wanted to minimize the *holes* in the sampled domain (which it could be shown to be a property that helps the convergence of the sequence) and minimize as well the *holes* in every lower-dimension projection of the sampled points. The particularly good distributions that fulfill those requirements are known as (t, m, s)-nets and (t, s)-sequences in base b.

To better understand them we need first to define the concept of s-interval in base b, which is a subset of I_s such as:

$$E_s^b = \prod_{j=1}^{s} \left[\frac{a_j}{b^{d_j}}, \frac{a_j + 1}{b^{d_j}} \right), \tag{2.14}$$

where a_j and d_j are non-negative integers, and $a_j < b^{d_j}$ for all j in {1, ...,s}.

Let be t and m two integers such as $0 \leq t \leq m$. A (t, m, s)-net in base b is defined as a sequence x_n of b^m points of I_s such that:

$$\textbf{Card } \mathbf{P} \cap \{x_1, \ldots, x_n\} = b^t \tag{2.15}$$

for all the elementary interval $\mathbf{P}$ in base b of hypervolume $\lambda(\mathbf{P}) = b^{t-m}$.

Given a non-negative integer t, a (t, s)-sequence in base b is an infinite sequence of points x_n such that for all integers $k \geq 0$, $m \geq t$ the sequence $\{x_{kb^m}, \ldots, x_{(k+1)b^m-1}\}$ is a (t, m, s)-net in base b.

Sobol in his article described in particular (t, m, s)-net and (t, s)-sequence in base 2. A more thorough description of all the formal properties of those particular sequences could be found in [42].

In order to perform the actual sampling during the modelization, it has been used the `saltelli` module from the `SALib` library, which performs sampling in an s-dimensional space following the Saltelli algorithm, which is a specific improved version of the Sobol algorithms oriented toward the parameter sensitivity analysis [36], [37].

2.5 Planar Section of a Polyhedron

As will be shown in section 3.1 a fundamental step for the functioning of the modelization is the planar section of a three-dimensional polyhedron. It turned out that there is no general rule to perform a planar section of a convex polyhedron with an arbitrary number of faces, respect to an arbitrary sectioning plane. Hence, I devised an algorithm to handle this task. In the case of a full intersection, the result of the sectioning process of a polyhedron is a polygonal surface, otherwise, it could be an empty set of points or a segment in case of particular tangency, but those two cases are not of interest to the model. This tool has been created from scratch by me in the preliminar design phase and also the implemented algorithm has been devised by me and chosen as the best option among other possibilities. It is not fruit of an extended and thorough formal analysis, hence it has not the prententious to be an extremely optimized algortihm. Nevertheless, this tool passed all the test I required and yielded the expected results.

Given a convex polyhedron with n vertices and a sectioning plane p, let V be the set of all the vertices and $f_p(\vec{x})$ the equation defining the plane. The algorithm is defined by the following steps:

1. Divide V in two subsets: A made of those vertices which lie above and B, made of those which lie below the sectioning plane. Like in 2.16:

$$\begin{aligned} A &= \{\vec{v} \in V \mid f_p(\vec{v}) \geq 0\} \\ B &= \{\vec{v} \in V \mid f_p(\vec{v}) \leq 0\} \end{aligned} \tag{2.16}$$

   If any of the two subsets tourns out to be empty the plane p does not intersect the polyhedron, and the section is empty. A and B are represented in different colors in Figure 2.5.

2. Detect, and *draw*, any possible line that crosses two points respectively from A and B. If n_A and n_B are the numbers of points above and below the plane then there

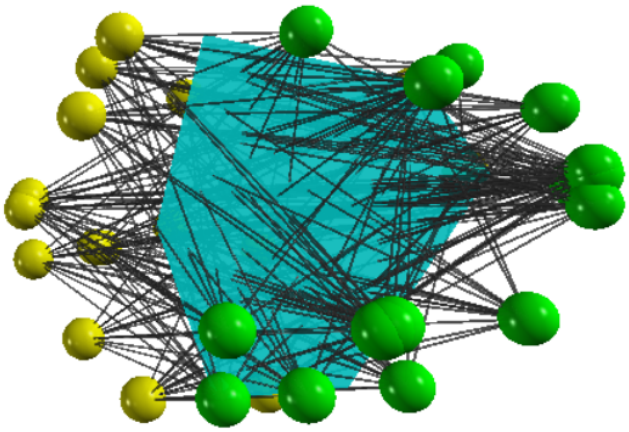


Figure 2.5: *In this picture is shown an example of the application of the algorithm for the planar section on a polyhedron. The vertices are divided into two groups, with different colors yellow and green. All the possible lines between any couple of vertices picked from the two classes are drawn in dark gray. In Turquoise the resulting planar section, obtained as the convex hull containing all the intersections between the lines and the plane.*

will be $n_A \times n_B$ possible lines. In Figure 2.5 all the lines between the two classes of points are drawn in dark gray.

3. Detect P, the set of all the points from the intersection between the $n_A \times n_B$ lines from the previous step and the sectioning plane p. All these points will lie on the same plane, within the boundaries of the polygonal section.

4. The final polygon is then yielded by computing the convex hull of the points in P. The convexity of the starting polyhedron in fact ensures the convexity of any section of the solid.

The result of the algorithm is, as just stated, a convex hull, which in geometry is defined as the smallest convex envelope or convex closure of a set of points. In 2 dimensions is the smallest convex polygon containing a certain set of points in a plane (In Figure 2.6 is shown the so-called "rubber band effect"), and in 3 dimensions it is the smallest convex polyhedron containing a set of points in the space.

In `Python`, the most convenient way to work with convex hulls was to use the sub-module `spatial.ConvexHull` from the `SciPy` library [45]. This module allows also a convenient way for plotting images with `MatPlotLib`, which is the point of reference for plotting and image formation in `Python`.

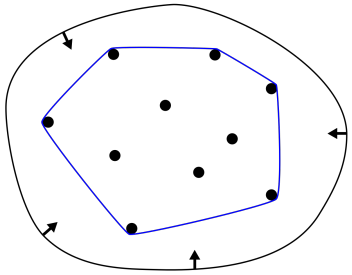
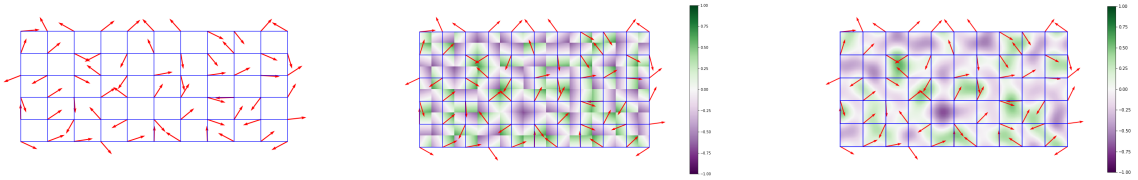


Figure 2.6: *Representation of the convex hull of a bounded planar set of points. This particular enclosure goes under the name of "rubber band effect".*



(a) ***(b)*** ***(c)***

Figure 2.7: *The three main steps of the algorithm to produce Perlin noise. In 2.7a the plane discretization and the assignment of a gradient vector to every node of the grid. In 2.7b the computation of the dot product with all the points inside the discretization and in 2.7c the interpolation of the values to create the final function.*

2.6 Perlin Noise

Perlin noise is a widely used form of noise in computer graphics, which mimics very well natural and smooth fluctuations around a constant value. It has been developed by Ken Perlin in 1983, and it is now the staple tool for giving texture to object in virtual modelization, often considered the *salt* of computer graphics texturization. The Perlin noise is a gradient-based algorithm defined on grid discretization of a n-dimensional space. The algorithm involves three subsequent steps:

1. The first step is to discretize the n-dimensional space in a regular lattice: the dimension of the grid will impact heavily on the scale of the noise. As in Figure 2.7a at every node of the grid is assigned a randomly oriented n-dimensional unitary gradient vector. This is the preliminary setup which will allow the computation of the actual noise function in every point of the space.

2. Given the candidate point $\vec{x}$ between the grid nodes onto which evaluate the noise there are 2^n nearest grid nodes. For each one of these 2^n nodes, it is evaluated the

distance vector from $\vec{x}$ as the offset between the two points. Then it is computed the dot product between every pair made of nearby gradient vectors and the offset vector. This operation should be thought of as made on every point in the lattice, as in Figure 2.7b, where at every point of the grid is represented just one of the $2^2 = 4$ series of dot products.

3. The final step is the interpolation between the 2^n series of dot products. To perform the interpolation usually is used a function with vanishing first degree (and preferably also second degree) derivative in correspondence of the 2^n grid nodes [3]. This means that the noise function will pass through zero at every node and have a gradient equal to the pre-computed grid node gradient. These properties give Perlin noise its characteristic spatial scale and smoothness.

In general, the final result of the algorithm is a smooth function with a random-like behavior that mimics really well an organic appearance, like in Figure 2.8, with fluctuation around the value 0, with amplitude $\in [0; 1]$. The surface in Figure 2.8 has been produced plotting in 3D the results of the function `pnoise2` from the library `noise`, which offers a tool for the production of different type of noise. In order to put in practical use this module some adjustment were required. The particular function `pnoise2` simply yields the value of the Perlin noise surface in correspondence to a single (x, y) point in the plane in a deterministic way. There was not the possibility to generate different whole Perlin surfaces every run. To overcome this limitation I made a vectorized[4] version of `pnoise2` able to evaluate the function over an arbitrary wide grid of points expressed as set of all the pairs of coordinates (x, y) of the grid's nodes. In this way a single call to the function is able to produce the entire surface covering the grid, in the form of a single `NumPy` 3D-array. Furthermore, to recover the possibility of generating always different surfaces as in a random generation, I inserted an offset cordinate (x_O, y_O), which moves in the plane the origin of the surface generation. This pair of offset coordinates then acts also as a *seed* in the generation, allowing to completely recreate previously generated material in a controlled way.

2.7 Style-Transfer Neural Network

Style-Transfer Neural Networks are common models, able of creating new hybrid images implanting the visual style from an image preserving the visual content of another image.

[3] Usually are used functions with a sigmoidal behavior, like any smoothstep function, which is a family of very common items in computer graphics.

[4] In Python the staple tool for scientific, number-cruncher computation is the `NumPy` library, which allows a fast, complete and efficient way to perform computation between number structure. The operation of trasnforming a function which acts on a single value (or pair of values) to a function able to perform on a suitable datastructure is called *vectorization*, and its the recomended way to proceed when handling numerical functions in Python.

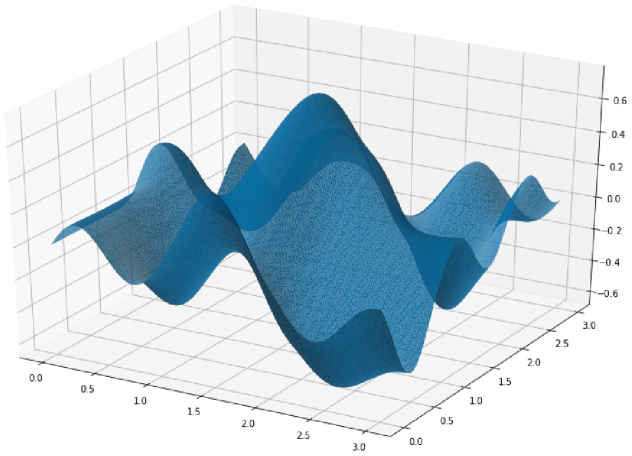


Figure 2.8: *Example of Perlin noise 2D funciton produced while working on this project. This surface offers a smooth variation around the value 0 with amplituded* $\in [0; 1]$.

The two images necessary for the algorithm are called *style* image S and *content* image C, and the resulting *styled* picture X, as in Figure 2.9.

There are many different tested and comparable architectures to compute this kind of algorithm. In my work I decided to use in particular the procedure described in [18], using the `PyTorch` ecosystem to implement the necessary code.

The backbone of the architecture is the VGG-19 network, which s a convolutional neural network 19 layers deep, as in Figure 2.10. This huge model has been pre-trained on over a million images from the ImageNet database [16], for the classification into over than 1000 classes of objects. As a result, the network has learned rich feature representations for a wide range of images. The best (and conceptually the only) way to load a pre-trained model is to load the ordered set of weights that define the network and to initialize an empty module with those values. This is the perfect start for creating a style transfer network, which requires a further and briefer training phase, to completely customize the network.

The key ingredient for finalizing the model is to insert some little but fundamental modifications and to extend the training on the pair of input images. This final training should be aware of the *concepts* of the visual style and visual content of the image, and the operation should try to preserve them both. This is usually done minimizing two new loss function, computed between the staring image and the produced image:

Content Loss

The content loss is a function that represents a weighted version of the content distance for an individual layer. The most commonly used function to evaluate the preservation of content between two images is the simple Mean Squared Error as

Figure 2.9: *Different examples of application of a style-transfer NN on the same content image, with different style images, from [18]. The original picture depicts the Neckarfront in Tbingen, Germany (TOP-LEFT). The painting used as style image shown in the bottom left corner of each panel are in clockwise order: • The Shipwreck of the Minotaur by J.M.W. Turner, 1805 • Femme nue assise by Pablo Picasso, 1910 • Composition VII by Wassily Kandinsky, 1913 • Der Schrei by Edvard Munch, 1893 • The Starry Night by Vincent van Gogh, 1889.*

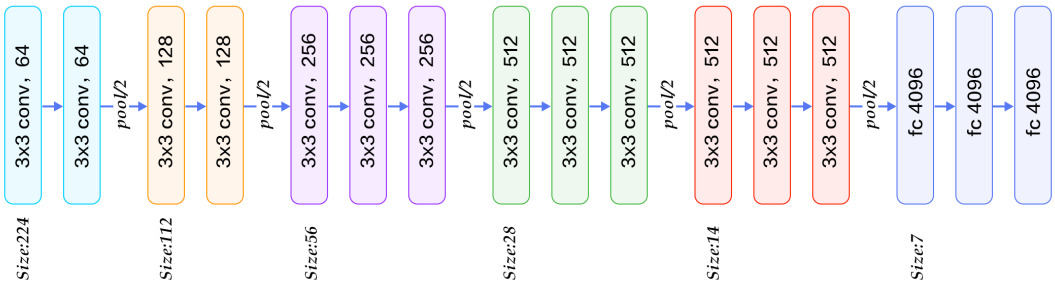


Figure 2.10: *The structure of the VGG 19 network. It is for its most a convolutional NN with 224×224 input size and with some downsampling layers which reduce the first two dimensions of the tensors along with the information flux of the network. At the very end of the architecture, there are three subsequent fully connected layers, responsible for the actual classification based on the features extracted from the previous layers.*

in equation (1.5). It can be computed between any couple of same-sized object, hence also between the results of the same feature maps on the images X and C at the same layer L:

$$L_{Cont} = \|F_{XL}F_{CL}\|^2 . \tag{2.17}$$

In order to evaluate this content loss, it is necessary to insert a custom transparent[5] layer directly after the convolution layer(s) that are being used to compute the content distance.

Style Loss

The concept of *style* loss function is the true novelty introduced by [18]. This loss function is implemented similarly to the content loss module, as it will act as a transparent layer in the network. The computation of the style loss requires in advanced the evaluation of the Gram matrix G_{XL} at a certain layer L. A Gram matrix is a result of multiplying a given matrix by its transposed matrix. In this case, the matrix to multiplicate is a reshaped version of the feature maps F_{XL}: $\hat{F}_{XL}$, a $K \times N$ matrix, where K is the number of feature maps at layer L and N is the length of any vectorized feature map F^k_{XL}. Furthermore, the Gram matrix must be normalized by dividing each element by the total number of elements in the matrix. The style distance is now computed using the mean square error between G_{XL} and G_{SL}:

$$L_{Style} = \|G_{XL}G_{SL}\|^2 . \tag{2.18}$$

After the appropriate insertion of the loss-function evaluator layers, one last piece for finalizing the model is the right choice of the gradient descent optimizer. As in [18] and according to the Deep Learning community the optimizer which suite best this role is the Limited Memory-BFGS [8],[39]. L-BFGS is an iterative algorithm in the family of quasi-Newton methods that approximates the Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS) using a limited amount of computer memory, and it is a popular choice when estimating parameters of a non-linear differentiable scalar function.

The final phase of the training process thus makes run the optimizer for hundreds of epochs on X, C, and S, and reduce the two loss functions values acting on the network's parameters. After the fine-tuning of the weights, the hybrid image is produced, as in Figure 2.9.

2.8 Working Environment

In this section I will briefly desribe the machine I used to develop my project and the working environment I built. All the work has been done on my personal computer,

[5] A transparent layer is a layer that performs some operations, like evaluating a function on its input, but returns as output an unchanged copy of its input.

mounting a `GNU/linux` operating system, in particular 18.04 LTS `Ubuntu` version. The computer mounts an Intel i7 core, 8 Gb of RAM beside an 2Gb NVidia 940MX GPU. All the Python libraries have been installed and harmonized in a virtual environment mounting `Python 3.7.6`. All the code produced during the development, the images, and the data produced have been collected in a devoted repository on GitHub [14], which is freely available.

As a conclusion for this chapter I will re-collect all the references to the different `Python` libraries I used during the development of this work:

`NumPy:` `NumPy` is the pillar of every scientific computation-oriented library. Is the most spread library for heavy multidimensional numerical computation, and it offers a broad variety of tools like random numebr generators, and pre-implemented linear algebra utilities [30].

`SciPy:` The `SciPy` library is one of the core packages that make up the SciPy stack. It provides many user-friendly and efficient numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, and statistics [45]. Two modules in particular form this libraries have covered an essential role in this project: the `SciPy.spatial.Voronoi` module for the computation of the 3D Voronoi decomposition, as mentioned in section 2.3, and the `SciPy.spatial.ConvexHull` module for the computation of 3D and 2D convex hulls (section 2.5).

`PyTorch:` `PyTorch` is a rich ecosystem of tools and libraries geared toward Machine Learning and Deep Learning. The application of the style-transfer NN described in section 2.7 has required the use of this framework [31].

`SALib:` The `SALib` is a library which collects many tools for the Sensitiviy Analysis of parameters. In particular the `SALib.saltelli` submodel was used for the quasi-random numerical sampling in a three-dimensional box, and it has been described in section 2.4.

`pnoise2:` `pnoise2` contains many tools for the production of specific types of noise. The module `noise` was tweaked for the production of two-dimensional Perlin noise surfaces, and it has been introduced in section 2.6.

`pyquaternion:` The `pyquaternion` library provides a framework for handling quaternions. It has been widely used in section 2.1 for the design of three-dimensional ramifications for handling multiple spatial rotations.

Chapter 3

~~Tissues Model~~ Tissue Models Development

The main goal of the present work, as stated before, is to recreate a three-dimensional virtual model of histological tissue as faithfully as possible and then, to perform planar sectioning on it to emulate virtually the traditional histological specimen preparation procedure. The creation of a model of such complex structures is definitely a high-level problem, and it has required a careful designing, made of subsequent stages of improvements. In section 3.1, I will describe all the necessary steps to create the model of a small region of pancreatic tissue, while in section 3.2 I will expose the steps I followed to build a model of dermal tissue. In section 3.3, instead, I will show the resulting synthetic images from the sectioning process performed on both the models and all the enrichments and processing necessary to give them the most realistic look I was able to recreate.

3.1 Pancreatic Tissue Model

The Pancreas is an internal organ of the human body, part of both the digestive system and the endocrine system. It acts as a gland with both endocrine and exocrine functions, and it is located in the abdomen behind the stomach. Its main endocrine duty is the secretion of hormones ike insulin and glucagon which are responsable for the regulation of sugar levels in the blood~~and the secretion of hormones, like insulin and glucagon. While, as~~. As a part of the digestive system instead it acts as an exocrine gland secreting pancreatic juice. The majority of pancreatic tissue has a digestive role, and the cells with this role form clusters (*acini*) around the small pancreatic ducts~~, and are arranged in lobes~~. The acinus secrete inactive digestive enzymes called zymogens into the small intercalated ducts which they surround, and then in the pancreatic blood vessels system [26]. In Figure 3.1 is shown a picture of the pancreas, with its structure and its placement

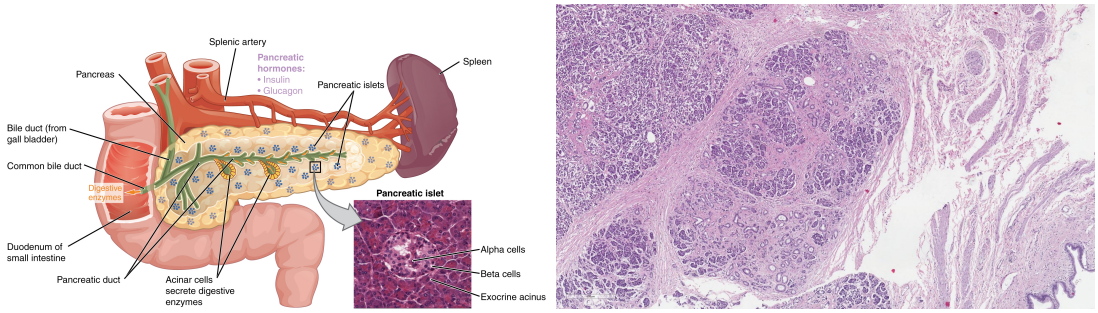


Figure 3.1: *(left) A picture of pancreas' structure in its phisiologiacl context. In this picture is clearly visible the macroscopic structure and the galndular organization at microscopic level, and how it reflects in the histological sample. (right) A real pancreatic tissue sample with H&E staining.*

in the human body.

All the tissue is actually rich in other important elements as the islets of Langerhans, and sporadic connective tissue all over the structure, which are clearly visible in the traditional histological specimens. In this first attempt of modelization from scratch this second layer of complexity has not been already considered, and the main focus has been to reflect only the main structural features on the virtual specimens. Given pancreatic tissue's organization the first features I decided to put emphasis on were: 1) The iterative (with a fractal-like behavior) ramification of blood vessels for the irrigation of glandular acinus, 2) The space-filling distribution of acinus in the tissue, in fact, we expect a homogeneous density in the organ and to not see *holes* at all inside it. In this section I will describe step by step all the process I followed to create the model of a portion of pancreatic tissue, and all the interesting pitfalls I overcame.

3.1.1 2D Ramification

The first step was took in two dimensions, and it was the choice of the right *structure* to emulate the ramification of blood vessels in pancreatic tissue. The choice fell on a particular parametric L-system, as the one shown in Figure 2.1, in section 2.2. This structure is made of an iterative bifurcation of gradually shorter segments, with an angle of $\pm 85°$ respect the main direction. For a start I added some features to give a more realistic look to the structure, which are all well represented in Figure 3.2:

- A progressive thickness of the bifurcation's segments, starting from a thick main branch that dwindles every junction. The idea is that the main blood vessel becomes gradually smaller becoming capillaries for single-cell irrigation.

- A progressive randomness in the angular deflection at every fork. Perfectly repeated angles are almost nonexistent in nature, so I decided to introduce an increasing

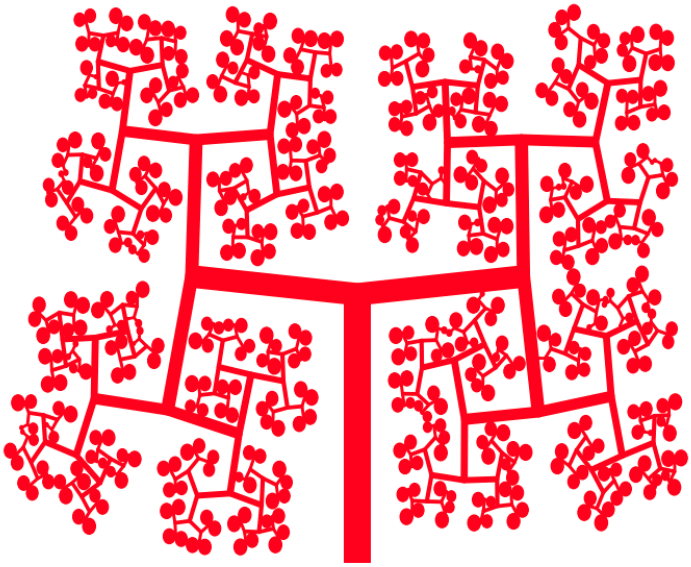


Figure 3.2: *The development of the simple ramification in Figure 2.1, with some features to give it a more realistic look, like progressive thickness, angular noise in bifurcation and spheres at free ends of the ramification. The image is made using the tools exposed in section 2.2.*

indetermination in the angle of bifurcation from the main branch to the free ends of the structures' branches.

- Spheres at the ends of each branch, which acts as glandular acini. The maximum radius is comparable to the length of the final segments.

- A mechanism to avoid self-superimposition between branches and spheres. After the insertion of noise, the cumulative effect on the final segments might lead different branches to intersect. This is clearly a paradoxical situation, as real tissues while growing naturally occupy the space in a gradual way.

To produce the specific image in Figure 3.2 I used a particular setting of the tool described in section 2.2, which have a greatly wider range of customization and could be used to create many other different structures to the need.

3.1.2 Expansion to 3D

The successive step I followed was to expand this structure in three dimensions and fill the space in each of the three directions. The idea to evolve the structure in Figure 3.2 is simply to twist of 90° the ramification at every junction point, in such a way to exit the previous belonging plane. However, putting into practice this development has not been

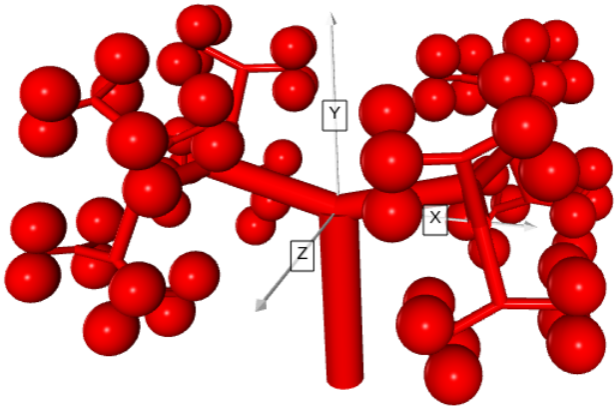


Figure 3.3: *The three-dimensional expansion of the 2D ramification in Figure 3.2.*

easy. The organization of the structure in a 3D space requires an appropriate system of reference for handling subsequent rotations in three dimensions. The best option for handling relative 3D rotations, often used in computer graphics and every kind of 3D modelization, are quaternions, as shown in section 2.1.

In this new structure, segments are replaced with cylinders, and circles are replaced with spheres. At every bifurcation to every cylinder are applied the following transformations:

- a contraction in its extensions, regulated by an adjustable parameter R.
- the usual deviation of $\pm 85°$ respect to the direction of the parent branch.
- a $90°$ specific rotation along the axis of its parent branch.

The result of this procedure is a 3D ramification like the one in Figure 3.3, in which we can recognize a good coverage of the space defined by the structure's boundaries and immediate relation with the 2D structure in Figure 3.2. It should be noted that, in the further refinements of the model from now on, there won't be present the progressive angular indetermination on the direction of branches. Although it is a feature already implemented and working, it requires efficient control to avoid reciprocal overlapping between elements to produce a realistic structure. This second element has not been already developed and it would certainly enrich the representative power of the model.

As for the 2D ramification the production of this structure has required the implementation of a tool for the 3D generation with a greatly wider power, able to produce almost any type of three-dimensional iterative structure after the right adjustment, and with a high degree of customization. It is necessary to mention the fundamental tool

which allowed me to accomplish this step of the development, which is the `Python` library `VPython`: a library for 3D graphics visualization. This library allows a convenient and powerful interface to draw many types of objects and to move them around in space, which has been priceless to orient my self in three dimensions while developing the model and to produce all the 3D images visible in this work.

3.1.3 Subdivision in Cells

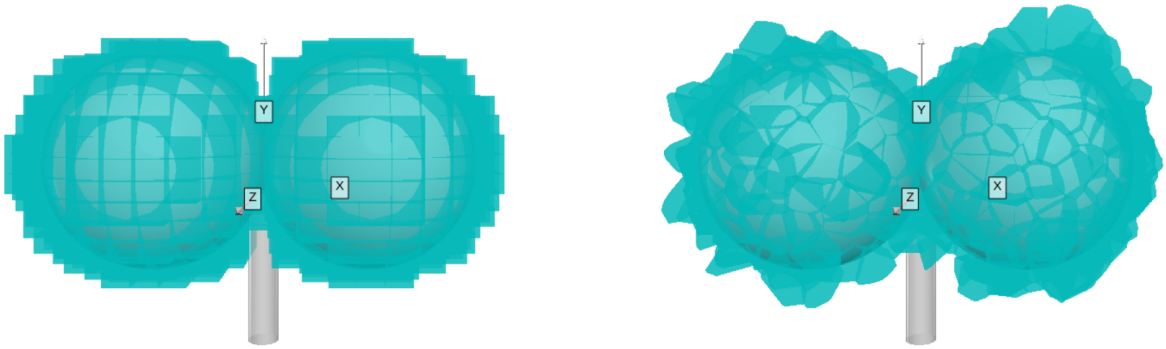
Once the 3D backbone of the pancreatic tissue blood vessels ramification system has taken shape, the next step was to embed all this structure in a spatial partitioning process, to create the subdivision into single cells. To perform this important task I used a 3D Voronoi decomposition, as shown in section 2.3. Depending on the choice of the starting points, the Voronoi tessellation could be an excellent item to recreate individual cells because it could guarantee some important properties: all the regions are convex, adjacent, with similar size and volume, with different shapes, and without holes. These have been chosen as the most significant properties to be reflected in the first modelization of cells.

As shown in section 2.3, the Voronoi decomposition strongly depends on the choice of the starting point. Points spread uniformly on a 3D regular lattice will produce a series of parallelepipeds repeated in the space. An example of uniform tessellation is shown in Figure 3.4a. On the other hand, a decomposition based on a quasi-random generated point can present all the good properties we mentioned before, including the diversity in shapes. In Figure 3.4b is shown an example of a Voronoi decomposition based on points sampled in a 3D with the Saltelli algorithm, in reference to section 2.4. Regardless of the points sampling technique, the boundaries of the sampling 3D box have been chosen to loosely contain the ramification.

There are tough some delicate considerations to be highlighted about the decomposition procedure. The first regards the most external pieces of the decomposition. Whilst the internal pieces are neatly bounded and defined, the most external layer instead is made on unbounded regions, which extend themselves to infinity. Those regions have clearly to be rejected, as it would be absurd for a cell to have an infinity volume. Typically those unbounded regions are resized in order to adhere to some limiting boundaries, with an operation known as *cropping*. In Figure 3.5 is shown an example of circular cropping in a 2D Voronoi decomposition: all the regions which intersect the circumference have to be resized.

The cropping operation in 3D is extremely complex, tough. Thus, a more simple and efficient, yet less elegant, technique has been used. Instead of resizing the regions which lie on the boundaries of the sampling region, those regions have directly been rejected. This process is really fast and it does not lead to any danger of representativity loss if the boundaries are loose enough and if the density of sampling is not too low.

The other important consideration regards the density of sampling points. Increasing



(a) *Regular lattice.* ***(b)*** *Sampling with Saltelli algorithm.*

Figure 3.4: *Comparison between two Voronoi decompositions. The first (left) is created from a regular lattice of starting points, and every piece is exactly equal to all the others, creating a regular subdivision of the space. The second (right) is created instead from a sampling made following the Saltelli quasi-random algorithm. The pieces are all different in shape, but they all have similar sizes and volumes. In this pictures in particular have been shown only the pieces of the tessellation which lie in correspondence to the boundaries of the spheres underneath. While watching this picture one should immagine the decomposition extended similarly in all the space around the ramification, within certain boundaries, which loosely contains the structure. This limitation was necessary to enhance the interpretability of the image.*

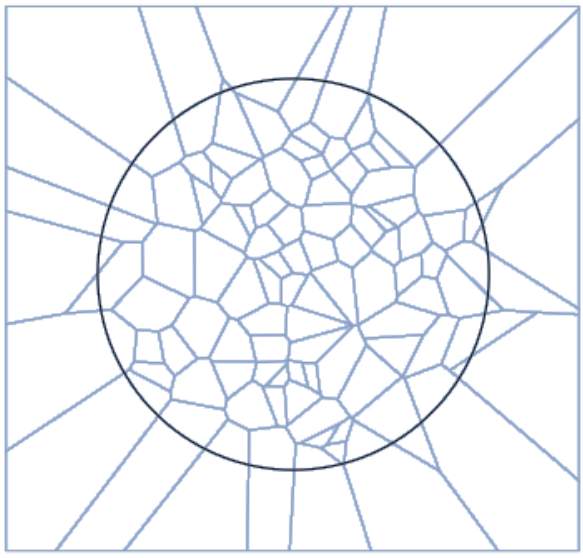


Figure 3.5: *Example of circular cropping in a 2D Voronoi decomposition: all the regions which intersect the circumference have to be resized.*

the number of points to be extracted from the same volume automatically the number of cells in the box will rise, and in contrast, their relative dimension will decrease. This is a key element of the model: a too rarified decomposition would not bee able to reflect the complexity of the structure underneath, but a too crowd decomposition on the other side would lead to an unrealistic dimension of the cells in the tissue. Furthermore, this parameter has a huge influence on the computing time necessary to generate the model and to process it for the sectioning as will be shown in section 3.3. In almost all the applications so far, the density parameter has been tuned by eye, with a trial and error procedure. Although, a more rigorous way to adjust this parameter would be to consider the average dimension of the cells and make some microanatomical considerations to define the correct relative dimensions. The measure of the volume[1] of the decomposition's regions is an accessible parameter, thus an easy way to estimate the average linear dimension of the cells can be to approximate all the cells to cuboid seeing their volumes as $V \approx L^3$. Averaging all the L measures an estimate $\hat{L}$ can be done. This average length may be compared to the length of the blood vessel ramification, allowing a good reference tool.

3.1.4 Cells Identity Assignment

The great power of creating all the models virtually is to know exactly the identity of every point in the structure. Although, This identity has to be reflected at the cellular level, assigning to every region a label. Imagining the Voronoi decomposition represented in Figure 3.4b extended to the entire box containing the ramification, good discrimination would distinguish three classes of cells: those which lie completely inside a sphere, those which lie completely outside a sphere, and those which lie on the boundaries of a sphere. In Figure 3.6 is shown a portion of the complete decomposition where the three classes of cells are reported with different colors: the internal cells in red, those on the boundaries in turquoise, and the external cells in gray.

In this particular case to find the relative position between every sphere in the structure and each cell it has been used a test on the proximity between the spheres' centers and the vertices of every polyhedral cell. If all the vertices of a region lie within a distance lower than the radius from the center of the same sphere then that region can be said to be an internal one. If none of the vertices lie within the radius distance from any center then that region is said to be external. In any other case, the region is said to be on the boundaries of some sphere, and this third label is assigned to it. As could be imagined the number of cells inside the volume can grow very quickly, and in the more rich ramifications also the number of spheres could be high. If we think that any polyhedron has a number of vertices of the order of 20/30 then it is clear that the number

[1] The volume is expressed in the same arbitrary length unit of measures used during the ramification structure. This allows a coherent reference tool.

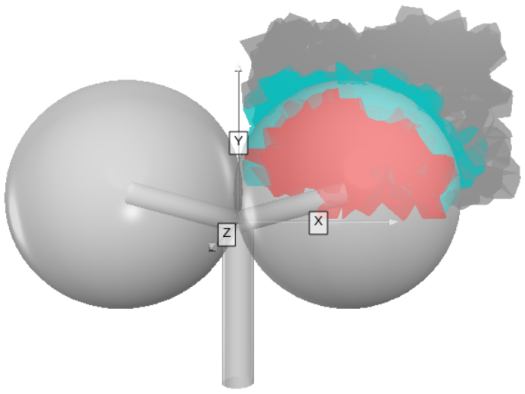


Figure 3.6: *Portion of the complete Voronoi decomposition, showing the three different classes of cell in three different colors: the internal cells in red, those on the boundaries in turquoise and the external cells in gray.*

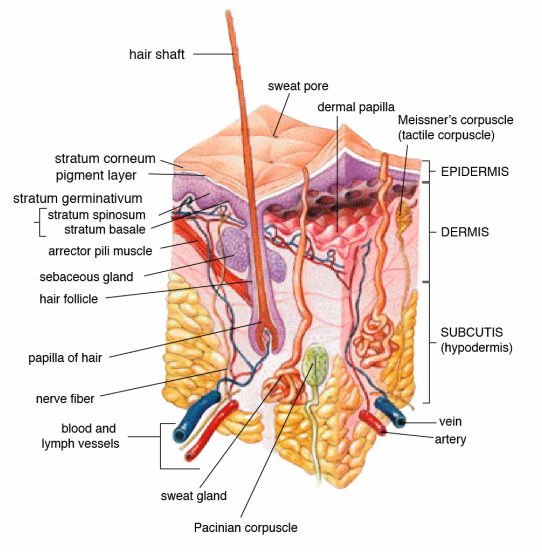
of distance evaluations could grow very quickly, requiring some relevant computational power in the more extended simulations. In order to optimize this computation, I decided to use a python implementation of a K-dimensional Tree, which is a space-partitioning data structure especially suited for fast and optimized computation of distances [6]. A K-d Tree is an algorithm that iteratively binary splits the space: every node of the three could be thought as a splitting $(k-1)$-hyperplane dividing the space into two semi-hyperspace. The result is an optimized algorithm for repeated distance evaluations. As for many other tools, in my code I used a pre-implemented module `KDTree` from the `Scipy` library.

This procedure of labeling the regions is completely customizable, and it should be adapted to the specific application. By the way, the principle will always be to perform some sort of spatial consideration respect to the primary structure and assign all the interesting labels accordingly to the cells in the volume.

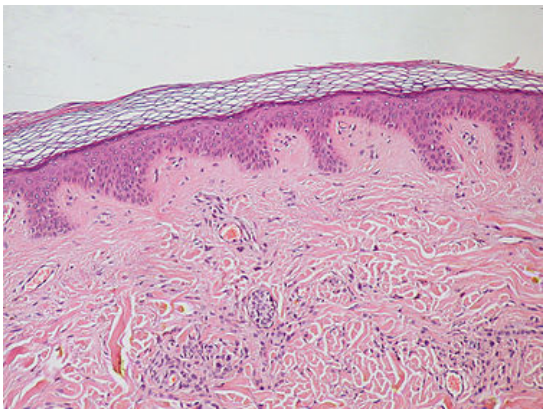
After labeling the cells in the decomposition the model is considered complete. Every enrichment to the structure should be reflected in some type of label for the cells, which are chosen as the fundamental unit in the model. As we will see in section 3.3 during the sectioning process in the produced image will be printed mainly the identity of the cells, hence any detail on a finer scale in the model would not be conveyed properly on the final image.

3.2 Dermal Tissue Model

Skin is the layer of soft, flexible outer tissue covering the body of a vertebrate animal, with the three main functions of protection, regulation, and sensation. Mammalian



(a)



(b)

Figure 3.7: *(left) Microanatomical description of a region of dermal tissue and all the interesting elements present in cutis, and subcutaneous layer. (right) An actual histological specimen from a sample of dermal tissue.*

skin is composed of two primary layers: the epidermis, which provides waterproofing and serves as a barrier to infection, and the dermis, which serves as a location for the appendages of skin. The epidermis is composed of the outermost layers of the skin. It forms a protective barrier over the body's surface, responsible for keeping water in the body and preventing pathogens from entering, and is a stratified squamous epithelium, composed of proliferating basal and differentiated suprabasal keratinocytes. The dermis is the layer of skin beneath the epidermis that consists of connective tissue and cushions the body from stress and strain. The dermis provides tensile strength and elasticity to the skin through an extracellular matrix composed of collagen fibrils, microfibrils, and elastic fibers, embedded in hyaluronan and proteoglycans.

The modeling of the dermal tissue has followed analogous schedule respect to the previous model, hence many procedures and considerations have been repeated. The main target for this model was to recreate the stratification of different specific tissues in the section of a dermal sample. As clearly visible in Figure 3.7b in a dermal histological specimen one can distinguish the lighter and wider region of proper *dermal* tissue, underneath a more shallow and darker region of *epidermal* tissue. It is very interesting the boundary between those two regions, which can be seen as an irregular and smooth surface, populated of dermal lobes. On the other side of the epidermal layer lies a layer of *keratin*, with a smooth and regular boundary surface. Keratin is a family of fibrous structural proteins known as scleroproteins, a key structural material for hair, nails, and

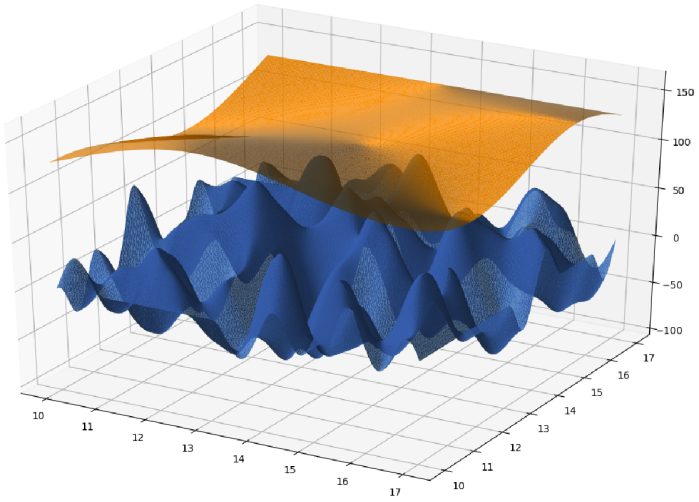


Figure 3.8: *Pictures of two different Perlin noise surfaces used to separate dermal from epidermal tissue (blue) and epidermal from keratine layer (orange). The two surface are made by the same Perlin noise function, but the latter is stretched and compressed in order to have a more regular behavior.*

the outer layer of skin. The upper white region in Figure 3.7b instead is the support for the samples to perform optical analysis, and has no histological meaning.

1) Stratificated Structure

In order to represent faithfully the stratification of different tissue layers, I decided to use one flat plane to separate epidermal tissue and the keratine layer and two boundary surfaces modulated by a Perlin noise function on different scales for the other two separations, as shown in Figure 3.8. As stated in section 2.6, after some easy customization the generation of different Perlin noise surfaces is easy and straightforward. By the way, to achieve the regularity and the smoothness of the orange surface in Figure 3.8 it was needed a horizontal stretching.

Following the scale of the image, the standard blue surface is created in a 7×7 square, while the orange surface has primarily been generated in a 1×1 square, and then has been stretched to cover the same 7×7 square, multiplying the values in its grid points respectively for the stretching factors $R_x = R_y = \frac{1}{7}$. In this primary structure, there are many important parameters defining the surfaces, like the distance between the two surfaes' average values and the amplitudes of the peaks and the valleys of the surfaces. In its standard version the Perlin noise covers the $[-1; 1]$ range, but with a simple multiplication for an amplitude factor A_S the values can be adjusted. Those particular values have been adjusted after some tries to recreate the proportions typical of a real specimen. To sum up, each one of the two surfaces is stored as a discretized three-dimensional array, or better

as an array of 3-tuples in the form $(x, y, f_{(x,y)})$, one tuple for every (x, y) node of the grid, while the discretization grid was the same for both the surfaces.

2) Subdivison in Cells

The subdivision in cells of the volume containing the structure has followed the exact same steps described in section 3.1, hence in this paragraph I will shortly resume the process. The first step is the definition of a suitable volume containing the structure. Then is the time for the generation of the decomposition's starting points according to a quasi-random number generation technique, as described in section 2.4. Afterward, the points are used as a base for the decomposition, and all the cells with undefined boundaries are rejected

3) Cells Identity Assignment

The identity assignment procedure instead has been customized for this particular application. In this model there are no *boundary* cells as the one lying on the spherical surfaces in the pancratic model, thus there is no need for a test on the position of each vertex of every cell. In this model, the starting point for every Voronoi region has been used as a reference, and its relative position respect to the boundary surfaces was the discriminating factor for assessing the identity. In order to perform a coherent test on the relative position between the regions' center and the boundaries surfaces the positions of all the centers had to be discretized on the same grid onto which were defined the surfaces. The comparison with the flat horizontal plane defining the boundary between epidermal tissue and the keratine layer instead was simply a test on the z coordinate of the point: $z = f_{(x,y)} \leq \hat{z}_{plane}$. The result of this procedure is that every region is assigned a label corresponding to the belonging tissue layer: D for dermal, E for epidermal, K for keratine, and V, which stands for *void* for the white empty space above the sample.

In this model, as in section 3.1, after the assignment of the identity to all the cells in the volume the modelization process is considered complete. Any sort of improvement and enrichment should be inserted during the structure designing phase and should be linked to an identity label.

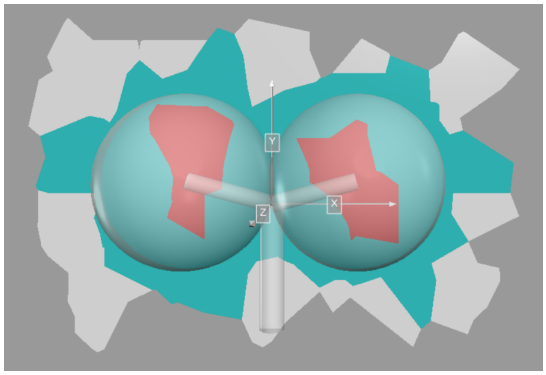
3.3 Synthetic Images Production

After the model is complete we have a three-dimensional representation of the tissue under study. The aim of the work is though to produce synthetic images from that structure, more precisely we want pairs of images composed of the synthetic-histological images and its related segmentation mask. The transition from 3D structure to a 2D image is the last step in the process, and it is inspired by the actual traditional technique for the preparation of the histological specimen, as described back in section 1.1.1. As the biopsy sample is treated and then sectioned with the microtome, the virtual model is sectioned in a random direction, producing an image representing the slice. This first image contains all the information of the section but its appearance is completely arbitrary and its look has nothing to share with a realistic sample. The original slice then acts perfectly as a segmentation mask, but some careful and dramatic makeover is needed to produce the final realistic looking image. In this section I will describe the general procedure to produce virtual slices from the two 3D virtual models described before in section 3 and the technique used to edit the images and give them the desired appearance.

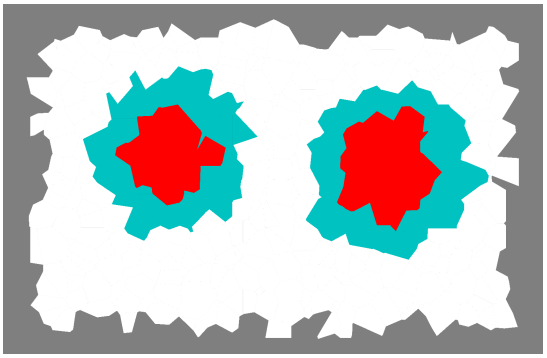
3.3.1 Sectioning Process

For any model created following the general procedure described in 3, even more so for the two particular models of pancreatic and dermal tissue, the sectioning process will be almost the same, and it will rely mainly on the algorithm for the general section of polyhedron described back in section 2.5. As stated before the models are essentially composed by labeled polyhedron spatially organized in a 3D volume. The ordered section of each polyhedron will yield all the polygons that shall be assembled in the final section. In Figure 3.9a is shown the three-dimensional representation of the section of a simple ramification, as the one in Figure 3.6. All the polygons that compose the section are drawn with the color correspondent to their label, following the same idea of Figure 3.6. In Figure 3.9b is printed the final result of the sectioning algorithm applied to the model, which will be the segmentation mask in the single pair of synthetic images. The colors in the produced slice match the colors used for the different identities in the 3D model.

The simplest, yet over-abundant, way to proceed is to create the model in its entirety and subsequently choosing the sectioning plane. Afterward, it is necessary to select only the regions that intersect the plane and section them all. Actually, the test on the intersection passes through the check on the relative position of the polyhedron's vertices respect to the sectioning plane: if all the vertices lie on the same semi-space then the intersection would be null and the polyhedron is not of interest for that particular section. This procedure is exactly the first step of the algorithm in 2.5, thus the filtering on the regions is actually made during construction for optimization. The alternative method could be to choose in the first place the direction of the sectioning plane, and in second



(a) *2D polygonal sections represented in a 3D environment.*



(b) *The fianl section image, produced directly on a planar picture, skipping completely the 3D representation.*

Figure 3.9: *In this Figure is shown the idea of the correspondence between the section of a simple structure in the space and the correspondent section. The correspondence is not perfect for representation requirments, in fact the two images even if very similar are produced with two completely different methods. At the left an image of 2D polygonal section embeded in a 3D space, made using 3D visualization tools. At the right a simple image produced printing the polygons in a planar picture. Printing 2D polygons in a 3D space is much more complicated than one would think using the same tool used to produce the other images like Figure 3.6 and 3.4. This choice has been done for the sake of the overall homogeneity in pictures style.*

place to generate the model's decomposition only in the volume adjacent to that plane. This method enables the sparing of a good amount of computation, without any negative impact on the final result. The only delicate step is the choice of a wide enough region of space around the plane, which doesn't compromise the representation.

As a guarantee for richness and diversification among the images, there is the need for some degree of controlled randomness in the sectioning process, for example in the determination of the sectioning plane direction. All the sectioning process is then based on a single starting *seed*, which determines the direction of the sectioning plane in a deterministic way, and all the rest of the model is generated as a consequence. In this way, all the possible angulations are equally probable and will be sampled in view of multiple applications of this process. In Figure 3.10 are shown two different sections, along two random planes on two simple ramified structures with four spheres.

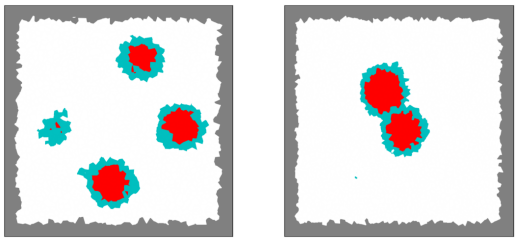


Figure 3.10: *Two different section, along two randon planes on two simple ramificated structure with four spheres.*

3.3.2 Appearance Makeover

After the application of the sectioning algorithm of the previous section, the image which will act as a segmentation mask is ready. The last and most complex task that remains is to transform the image and to give it a realistic look. I tried many different transformations, more or less complicated, and there was not a final decision on which is the best blend of them. In this section, I will describe them as an arsenal of possibilities and show their impact on the images.

Color Palette

The first correction to do to the images will inevitably be a change in the colors of the image. Gray, Turquoise, and Red are the perfect choice for label-colors but act poorly as physiological colors. In Figure 3.11 is shown an example of an image produced re-mapping the colors to a new palette, inspired to the coloring of the real specimen in Figure 3.1 and 1.2, given by the traditional hematoxylin and eosin staining process.

Figure 3.11: *Example of images produced re-mapping the colors with a color palette inspired to a real H&E stained histological sample.*

Nuclei Projection

Another fundamental processing needed was the projection of cells' nuclei on the image. Usually, nuclei are clearly visible in histological samples and guide the

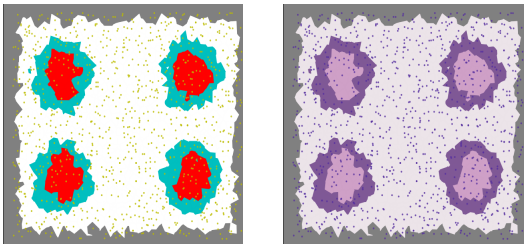


Figure 3.12: *Nuclei projection on the image: (left) in yellow in the segmentation mask and (right) in purple in the image under makeover.*

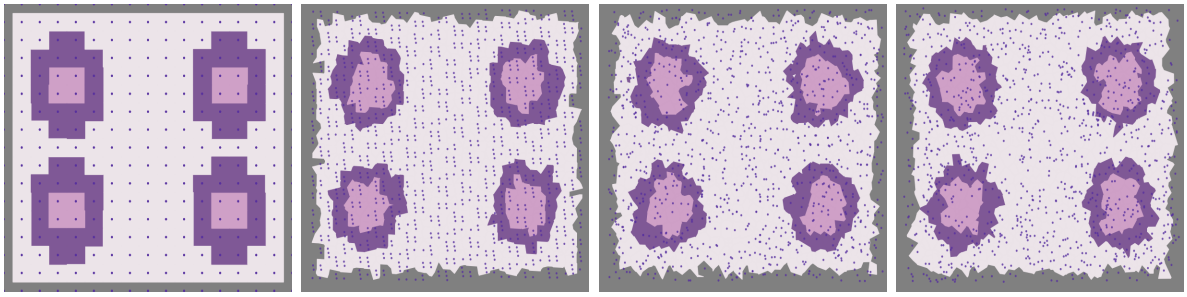


Figure 3.13: *Four different sections produced with the same density on cells but with four different method for the sampling of starting decomposition's ponits (from left to right): • points sampled on a regular lattice, • sampling following a simple recursion sequence as the one in equation (2.12), • following the saltelli algorithm, • following a fully-random distribution.*

analysis allowing to detect individual cells in the specimen. As a reference for the nucleus position the starting point of every polyhedral region has been used and projected on the sectioning plane as a little dark circle. The diameter of those circles as been chosen to be a submultiple (10%) of the linear estimated dimension $\hat{L}$ of the cells in the decomposition[2].

Nuclei projection, among the other things, is an excellent tool to perceive the different effects obtained with different choices of quasi-random distributions or fully-random distributions (with reference to section 2.4). The different impact on the overall image is huge, and it really changes the overall sense of the image. In Figure 3.13 are reported four different sections, produced with the same density on cells but with four different methods for the sampling of starting decomposition's points.

Boundaries Projection

[2]Following the same logic of step 3 of section 3.1.

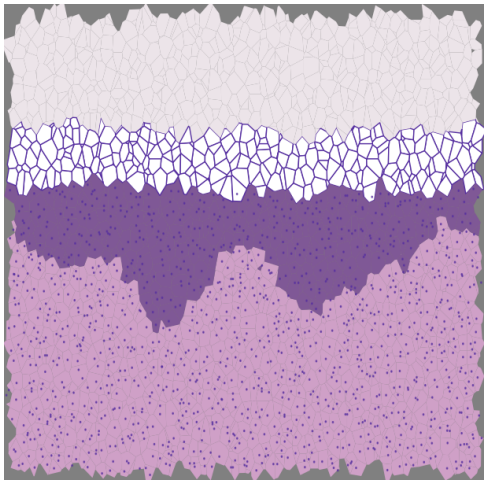


Figure 3.14: *Example of images produced re-mapping the colors with a color palette inspired to a real H&E stained histological sample.*

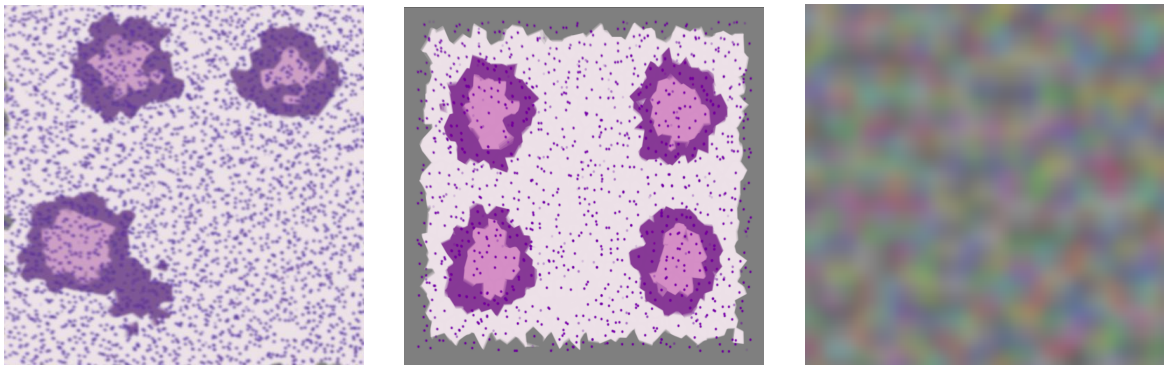
On the same wave of the previous tool, another operation that can help the appearance of an image is the projection of the boundaries of each (or just a part) of the polygonal sections. The drawing can be clearly tuned and customized depending on the specific necessities. In Figure 3.14 is shown an example of a section on the dermal tissue model in 3.2, in which the boundaries of all the cells have been lightly marked, and the boundaries of the cells in the keratine layer have been heavily marked instead.

Blurring Effects

In all the images produced so far the boundaries between polygonal sections are perfectly sharp and without any smudge. To give a more realistic feeling to those pictures I tried different forms of blurring. As a first try, I applied a Gaussian blurring filter, which is an extremely common blurring operation in computer vision, which consists of a simple discrete convolution with a 2D Gaussian kernel. The effect is a regular and diffuse blur all over the image, as in Figure 3.15a. The second blurring effect I implemented was based instead on the averaging of parallel and adjacent slices on the same model. This method is inspired by the real sectioning technique (section 1.1.1), in which every slice is not an infinitesimal layer of matter, but a finite sample, which suffers from mechanical dragging during the process. The idea is that the average between three or more slices equally spaced above and below the *main* slice should recreate a realistic blurring effect. An example of an image produced with this process is shown in Figure 3.15b.

Perlin RGB Noise

A further attempt to give visual texture to the image was done using again the



(a) *(b)* *(c)*

Figure 3.15: *The two blurring effects used in this work: (left) A standard Gaussian-filter blur and (center) a specific blur introduced averaging adjacent parallel slices on the same image, (right) an example of RGB color noise built joining three different Perlin noise surfaces, one for each color channel.*

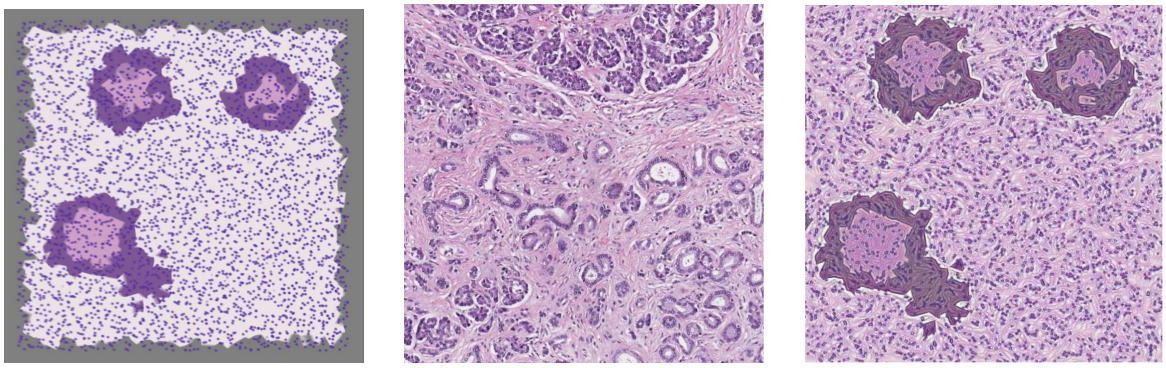
Perlin noise, described in section 2.6. The idea is to create some fluctuation among the color channels of the image around the sharp values of the image produced by the sectioning algorithm. From a practical point of view, I created three different and independent Perlin noise surfaces, one for every color channel (Red, Green, and Blue), and added them to create an RGB noise on the image. An example of the resulting image is shown in Figure 3.15c.

Style Transfer

This last tool I will describe is the most sophisticated so far. It consists of the application of a style-transfer neural network (STNN) on the image obtained through the sectioning process, for the implantation of the visual texture from a real sample of the corresponding tissue. Style-transfer NNs, and their functioning, have been described in detail in section 2.7, and here I will cover just the particular applications on the two type of section produced.

The first manipulation I report is the one on a section from the pancreatic tissue model. The image of which to conserve the visual content is a section with some simple pre-processing picked from the ones described before (Figure 3.16a): a more accurate color palette, the projection of nuclei, and the average on five adjacent slices. The image from which to pick the style, thus the visual texture, is a portion of an actual histological sample of the pancreas, and it is shown in Figure 3.16b. The application of the STNN yields a hybrid image, shown in Figure 3.16c. The second application was made on a section on dermal tissue. The three content, style and styled images are shown respectively in Figure 3.17a, 3.17b, and 3.17c.

In Figures 3.16, and 3.17 are reported the best results among all the different tests



(a) *(b)* *(c)*

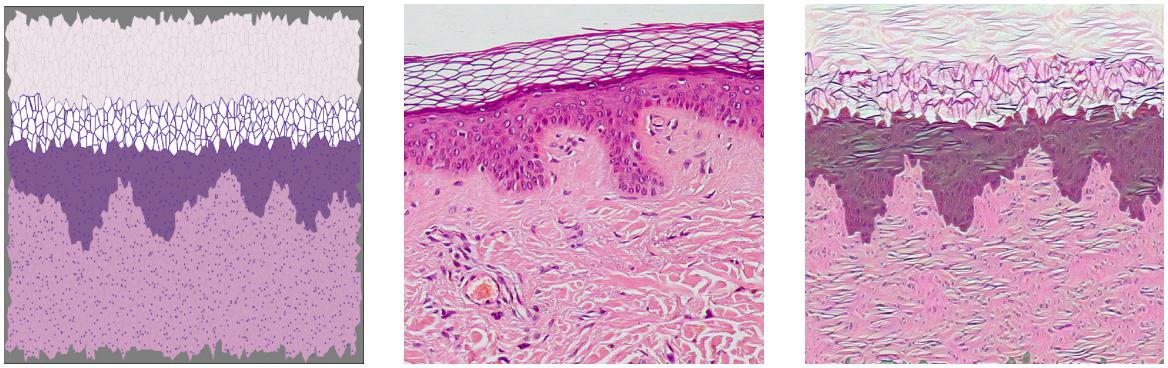
Figure 3.16: *Application of the style-transfer NN on a section of the pancreatic tissue model: 3.16a the content image, 3.16b the style image, 3.16c the hybrid resulting image.*

made on the sections. I made different tries on the same image with different processing before the manipulation with the STNN, to see the impact of the different adjustment on the resulting styled image. It turned out that the presence of nuclei is essential to give a homogeneous texture to the image and avoid unrealistic artifacts. On the other hand, the choice of the color palette has a way lighter effet than what one would think: the model yields almost the same result with a grey-levels image or with any other palette.

It interesting to notice the timing cost of this style transfer operation. While all the other manipulation described in this chapter requires a very short time (seconds) to be applied, and are in practice *instantaneous*, the transfer of style is a way more robust operation, which implies the finalization of the training of a pre-trained neural network. On a computer with the technical specification described in section 2.8 this operation instead took minutes, which is a time two full orders of magnitude greater.

It should be noted that the presented results are obtained from the application of a pre-trained STNN model. The development of a specialized model for histological texture transfer could improve extremely the ability to produce realistic images, way further the present results.

One single complete application of the process consists then in the generation of a tissue model, in the sectioning along a random section plane, and in the processing of the image, in order to produce the pair of ground-truth image and the synthetic histological image. The target is to apply over and over this process to collect the necessary amount of images and constitute an entire dataset. An important feature to have for the process is thus a complete automatization, in order to scale up the



(a) *(b)* *(c)*

Figure 3.17: *Application of the style-transfer NN on a section of the dermal tissue model: 3.17a the content image, 3.17b the style image, 3.17c the hybrid resulting image.*

generation of images, possibly even in parallel computation. For this purpose, I created a pipeline workflow interface for the image generation, with an automatized harmonization of every piece of the process. The generation now requires just to fill a configuration file in which writes all the specific characteristics of the images: the type of structure, its features, the desired processing on the images, and eventually the random seeds for a supervised generation. In Figure 3.18 is reported a small scale example of a dataset produced with multiple automatized applications of the generation tool on a ramified structure inspired to a pancreatic tissue model. It is clear the correspondence between segmentation mask and synthetic histological images, and the diversification given by the supervised randomness on the generation.

The actual tool used for the set up of a working pipeline was the `Snakemake` [24] workflow management system, which is a python-based tool to create reproducible and scalable data analyses.

3.3.3 ~~Alternative Works on Synthetic Histological Images Generation~~

~~As a conclusion to this chapter, I want to make an overview of some other interesting works on the generation~~

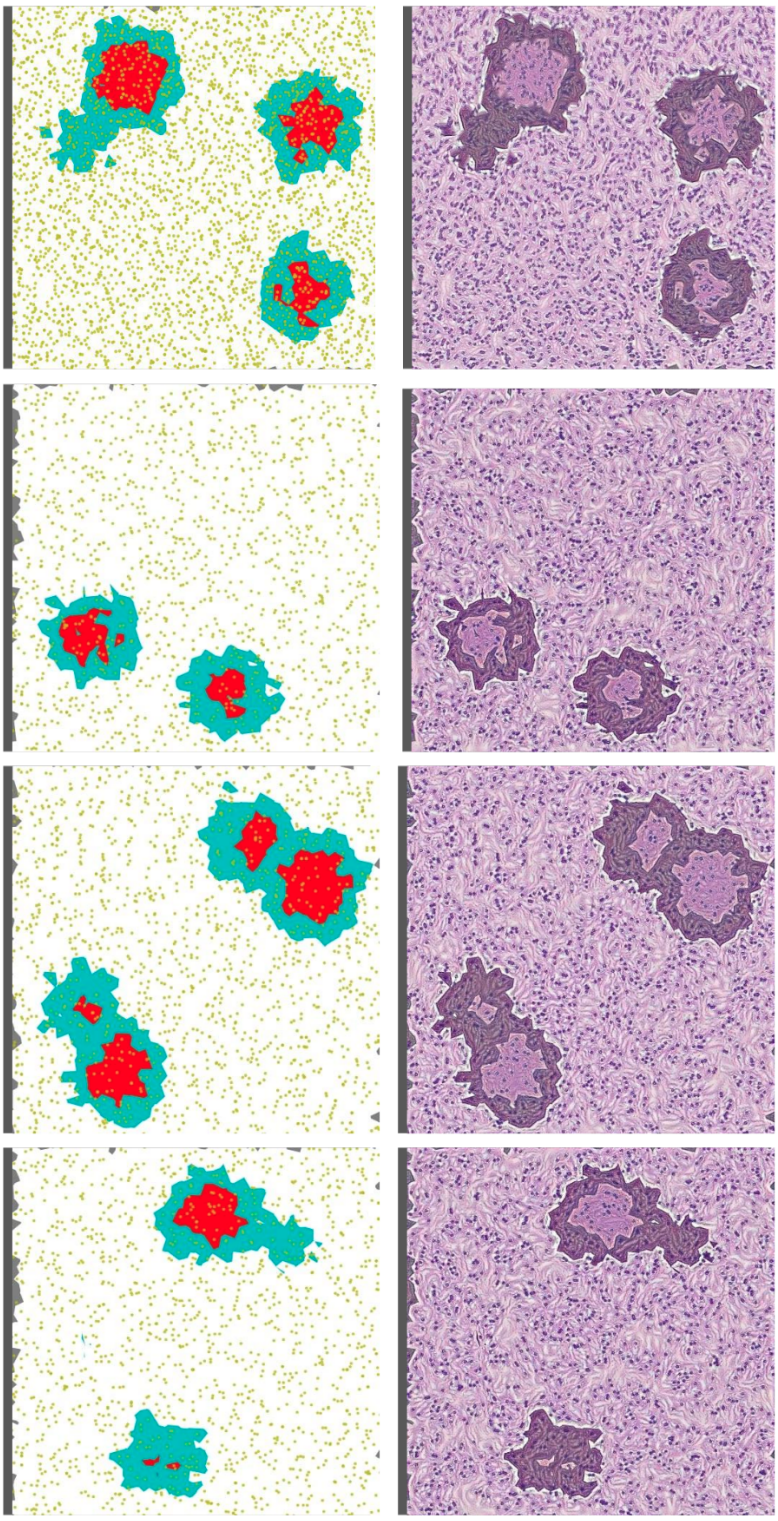


Figure 3.18: *Small scale example of dataset produced with multiple automatized applications of the generation tool on a ramificated structure inspired to a pancreatic tissue model.*

Conclusions

In this project, I face the problem of synthetic histological ~~images, which have followed completely different paths and strategies.~~

~~The first work I want to cite is a work from Ben Cheikh *et al.* from 2017 [9]. In~~ image generation for the purpose of training Neural Networks for the segmentation of real histological images. The manual analysis of histological specimen is a complex, time consuming, and expensive task and nevertheless it is a pillar of countless diagnostic techniques. Any form of support for this procedure hence is welcome and endorsed by the health-care system. In particular, in this work, ~~they present a methodology for the generation of synthetic images of different types of breast carcinomas. They propose a method completely based on two-dimensional morphology operation, as successive image dilations and erosions. With the modulation of a very restricted number of parameters, regulating the abundance of the objects, their distribution in the image, and their shapes they are able to reconstruct realistic images reflecting different histological situations. In Figure 2 is shown an example of generated material besides a real histological H&E stained sample. Starting from a generated segmentation mask which defines the *tumoral pattern* the production of the synthetic image passes through successive steps, as the generation of characteristic collagen fibers around the structure, the injection of all the immune system cells, and some general final refinements.~~ I focus on the problem of histological specimens segmentation. The most advanced algorithms for image segmentation are based on Deep Learning and requires the training of extensive and complex neural networks. One of the toughest hurdles to overcome for the training of those NN is the abundance and the quantity of pre-labeled examples of segmentation on real histological samples. The collection of hundreds of hand-labeled histological samples, with pixel-level precision, is virtually impossible. This work thus proposes a methodology to generate, in a completely automatic way, synthetic pre-labeled histological-like images, that can be used as training material for a NN.

~~Example of generated tumoral pattern (left), which acts as segmentation mask, of generated image (center) and a real example of the tissue to recreate. From [9].~~ The method I propose consists of the recreation of the traditional histological specimens' preparation, and it is based on the sectioning of a 3D virtual model of a region of histological tissue. The virtual 3D model of a region of a particular type of human tissue

is built after physical and physiological considerations. The model is then subject to a virtual sectioning operation, which yields the synthetic sampling of the virtual tissue in which the histological identity of every pixel is perfectly known. This first image will act as a segmentation mask for a second, realistic image. In fact, on top of this first image are applied several aesthetical processing and refinements and the final product is the synthetic histological-like image. The pair made of the two images is then perfect for the supervised learning of a NN oriented toward the segmentation of histological images. The production of each pair of images is completely automatic and it does not require the intervention of any human operator, it is thus a scalable process that can produce a great abundance of images. The quality of the images is directly connected to the richness and the quality of the model. The perfect modelization of a region of tissue, let's say human pancreatic tissue, is by far out of reach for this, hence the richness and the fidelity of the produced images are inevitably lower than the real sample. Nevertheless, the quality of the produced images is sufficient to perform the preliminary phase of the training of a NN following a training strategy known as curriculum learning. This learning process consists of giving the NN a copious quantity of lower complexity level example in the first instance, reserving the few and sophisticated real hand-labeled histological samples for the finalization of the training.

The first chapter of this thesis is devoted to the contextualization of the present work. It is offered a description of how the histological samples are obtained after a tissue biopsy and how the digitalization process of the images works. The reader is then introduced to the framework of Deep Learning, its fundamental aspect, and components. The concept of neural network is exposed and it is given the general idea underneath the training of a DL-based model. The chapter comes to an end with a general introduction to the segmentation task in computer vision, and how it is currently tackled with state-of-the-art algorithms.

The second ~~work I want to mention is base on a DL-base technique, which approaches synthetic image generationusing a specific cGAN architecture inspired to the "U-net" [38] model described back in Figure 1.13 in section 1.3.1. This model works with Ki67 stained samples of breast cancer tissue~~chapter collects all the details of every less common technical tool I used during the design of this project. A brief theoretical introduction is proposed for every item besides the thorough description of its practical use. Some of the arguments touched by this chapter are quaternions, quasi-random number generation, Voronoi decomposition, style transfer neural networks. In this chapter, a section is devoted to the description of a general methodology for computing the 2D section of an arbitrary three-dimensional polyhedron. The algorithm here described has been devised and implemented all by my self, and then inserted in the workflow of the project. As a conclusion for this chapter, I describe the working environment I built for developing this project and I mention all the code libraries I employed in my work.

The third chapter is the center of this work, and it ~~is able to generate high-fidelity images starting from a given segmentation mask. Those starting segmentation masks~~

tough are obtained through the processing of other real histological samples, via a nuclei-detection algorithm. The differences between real and synthetic samples are imperceptible, and the material generated in this work has effectively fooled experts, who qualified it has indistinguishable from the real one. In Figure 3 an example of a real image, a generated one, and their corresponding segmentation mask.

Example of generated tumoral pattern (left), which acts as segmentation mask, of generated image (center) and a real example of the tissue to recreate. From [38].

contains the description of all the design choices, and the steps I followed for the development of the two human tissue models I propose: the first of pancreatic tissue and the second of dermal tissue. The development has required the harmonization of many different technical aspects and mathematical tools. The first section is then dedicated to the description of the pancreatic tissue model, which passes through different steps: from a two-dimensional ramification taken as inspiration for the behavior of blood vessels to the complete three-dimensional model, with its subdivision in labeled cells. The second section is occupied by the description of the dermal tissue model, following the same spirit. The third section instead contains a thorough description of the method to perform the sectioning onto a virtual model and how to process the resulting images. It is necessary to perform several alternative processing and adjustment to achieve the desired aspect both for the segmentation mask image and for the histological-like image after the sectioning process.

The method for the generation of datasets of synthetic images I propose in my thesis work is a self-supporting project and it is formally consistent. By the way, there are many possibilities for improvement and enrichment for the project. One first aspect to strengthen could be the richness of the models: adding more elements in the structure, and refining their representation of the tissue at the cellular level. This would lead to a better quality of the synthetic images, that would assist the training of NN in more and different applications. Another aspect that lends itself to improvements in the development of a dedicated style transfer NN targeting the histological texture transfer, which could lead to interesting signs of progress in the quality of image generation. There is also the intention to perform an actual attempt of NN training on the images produced with this process. This would complete conceptually the idea underneath the project and would be an excellent opportunity to detect weaknesses and to draw up possible lines of development. The repeated application of the generation method would allow the building of entire datasets suitable for the training of DL-based models.

Bibliography

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018.

[2] Salah Alheejawi, Mrinal Mandal, Hongming Xu, Cheng Lu, Richard Berendt, and Naresh Jha. Deep learning-based histopathological image analysis for automated detection and staging of melanoma. In *Deep Learning Techniques for Biomedical and Health Informatics*, pages 237–265. Elsevier, 2020.

[3] J. Alsayednoor and P. Harrison. Evaluating the performance of microstructure generation algorithms for 2-d foam-like representative volume elements. *Mechanics of Materials*, 98:44 – 58, 2016.

[4] Hani A Alturkistani, Faris M Tashkandi, and Zuhair M Mohammedsaleh. Histological stains: A literature review and case study. *Global Journal of Health Science*, 8(3):72, June 2015.

[5] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 4148, New York, NY, USA, 2009. Association for Computing Machinery.

[6] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509517, September 1975.

[7] R. W. Brockett. Robotic manipulators and the product of exponentials formula. In P. A. Fuhrmann, editor, *Mathematical Theory of Networks and Systems*, pages 120–129, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[8] C. G. BROYDEN. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 03 1970.

[9] Bassem Ben Cheikh, Catherine Bor-Angelier, and Daniel Racoceanu. A model of tumor architecture and spatial interactions with tumor microenvironment in breast carcinoma. In Metin N. Gurcan and John E. Tomaszewski, editors, *Medical Imaging*

2017: Digital Pathology, volume 10140, pages 73 – 80. International Society for Optics and Photonics, SPIE, 2017.

[10] Anna Choromanska, Benjamin Cowen, Sadhana Kumaravel, Ronny Luss, Mattia Rigotti, Irina Rish, Brian Kingsbury, Paolo DiAchille, Viatcheslav Gurev, Ravi Tejwani, and Djallel Bouneffouf. Beyond backprop: Online alternating minimization with auxiliary variables, 2018.

[11] Siddharth Singh Chouhan, Ajay Kaul, and Uday Pratap Singh. Image segmentation using computational intelligence techniques: Review. *Archives of Computational Methods in Engineering*, 26(3):533–596, February 2018.

[12] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[13] Angel Cruz-Roa, Ajay Basavanhally, Fabio Gonzlez, Hannah Gilmore, Michael Feldman, Shridar Ganesan, Natalie Shih, John Tomaszewski, and Anant Madabhushi. Automatic detection of invasive ductal carcinoma in whole slide images with convolutional neural networks. *Progress in Biomedical Optics and Imaging - Proceedings of SPIE*, 9041, 02 2014.

[14] Alessandro d'Agostino. Dataset generation for the training of neural networks oriented toward histological image segmentation, april 2020.

[15] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[17] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[18] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] Johannes Kster and Sven Rahmann. Snakemakea scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 08 2012.

[25] Aristid Lindenmayer. Mathematical models for cellular interactions in development ii. simple and branching filaments with two-sided inputs. *Journal of theoretical biology*, 18(3):300–315, 1968.

[26] Daniel. Longnecker. Anatomy and histology of the pancreas, 2014.

[27] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey, 2020.

[28] Muhammad Niazi, Thomas Tavolara, Vidya Arole, Douglas Hartman, Liron Pantanowitz, and Metin Gurcan. Identifying tumor in pancreatic neuroendocrine neoplasms from ki67 images using transfer learning. *PLOS ONE*, 13:e0195621, 04 2018.

[29] Arild Nkland and Lars Hiller Eidnes. Training neural networks with local error signals, 2019.

[30] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[32] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143195, 1999.

[33] Prabu Ravindran, Adriana Costa, Richard Soares, and Alex C Wiedenhoeft. Classification of cites-listed and other neotropical meliaceae wood images using convolutional neural networks. *Plant methods*, 14(1):1–10, 2018.

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[35] Juan Rosai. Why microscopy will remain a cornerstone of surgical pathology. *Laboratory Investigation*, 87(5):403–408, April 2007.

[36] Andrea Saltelli. Making best use of model evaluations to compute sensitivity indices. *Computer Physics Communications*, 145(2):280 – 297, 2002.

[37] Andrea Saltelli, Paola Annoni, Ivano Azzini, Francesca Campolongo, Marco Ratto, and Stefano Tarantola. Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index. *Computer Physics Communications*, 181(2):259 – 270, 2010.

[38] Caglar Senaras, Muhammad Khalid Khan Niazi, Berkman Sahiner, Michael P. Pennell, Gary Tozbikian, Gerard Lozanski, and Metin N. Gurcan. Optimized generation of high-resolution phantom images using cGAN: Application to quantification of ki67 breast cancer images. *PLOS ONE*, 13(5):e0196846, May 2018.

[39] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[41] Sandro Skansi. *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st edition, 2018.

[42] I.M. Sobol. Uniformly distributed sequences with an additional uniform property. *USSR Computational Mathematics and Mathematical Physics*, 16(5):236 – 242, 1976.

[43] I.M Sobol. Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. *Mathematics and Computers in Simulation*, 55(1):271 – 280, 2001. The Second IMACS Seminar on Monte Carlo Methods.

[44] M Titford. The long history of hematoxylin. *Biotechnic & Histochemistry*, 80(2):73–78, 2005.

[45] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde,

Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[46] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1908:102 – 97.