UNIVERSITAT DE BARCELONA

Bachelor's Thesis

Double bachelor's degree in Mathematics and Computer Science

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

ADVANCED SEMANTIC SEGMENTATION USING DEEP LEARNING

Author: Albert Sallés Torruella

Director:	**Dr. Simone Balocco**
Developed in:	**Departament de Matemàtica aplicada i anàlisi**

Barcelona, **January 22, 2022**

Abstract

Intravascular Ultrasound (IVUS) is an intra-operative imaging modality that facilitates observing the vessel wall structure of the human coronary arteries. Segmentation of the arterial wall from IVUS images is crucial for the analysis of the wall's characteristics and for 3D reconstructed models of the artery. The aim of this project is to learn, study, develop and evaluate a model capable of performing semantic segmentation on medical images using a small IVUS dataset. The dataset used contains 435 pullbacks from 10 different patients acquired by 20 MHz probes. Our proposed model uses the InceptionResNet architecture pretrained with the ImageNet dataset as the encoder. The model is then connected like a U-Net with a decoder which is trained using the images. We have explored different approaches in data augmentation and the best results were given by rotations up to 90 degrees, vertical & horizontal flips and zooming. Among the results, we obtained an average of Dice similarity coefficient, or Dice Index, of 0.75 for the media and 0.92 for the lumen, with pretty low variance. However, the Hausdorff distance (HD) from results show that some misclassified sections appear in average 0.51 and 0.67 mm away from the ground truth class, for the lumen and media respectively. Finally, the Jaccard Measure for the lumen is quite high: 0.85; whereas for the media is 0.60, which is pretty low. The results found in this project present a good start on how to implement a solution for the problem explained.

Resum

L´ecografia intravascular (Intravascular Ultrasound, IVUS) és una modalitat d´imatge intraoperatòria que facilita l´observació de l´estructura de la paret de les artèries coronàries humanes. La segmentació de la paret arterial a partir d'imatges IVUS és crucial per a l'anàlisi de les característiques de la paret i els models 3D reconstruïts de l'artèria. L'objectiu d'aquest projecte és aprendre, estudiar, desenvolupar i avaluar un model capaç de fer segmentació semàntica en imatges mèdiques utilitzant un petit conjunt de dades IVUS. El conjunt de dades utilitzades conté 435 extraccions de 10 pacients diferents adquirides per sondes de 20 MHz. El nostre model proposat utilitza l'arquitectura InceptionResNet preentrenada amb el conjunt de dades ImageNet com a codificador. A continuació, el model es connecta de la mateixa forma que una U-Net amb un descodificador que s'entrena fent servir les imatges. Hem explorat diferents enfocaments en l'augment de dades i els millors resultats s'han donat utilitzant rotacions de fins a 90 graus, flips verticals i horitzontals i zoom. Entre els resultats, hem obteningut una mitjana del coeficient de similitud de Dice, o índex de Dice, de 0,75 per la

media i de 0,92 per al *lumen*, amb una variància força baixa. No obstant això, la distància de Hausdorff (HD) dels resultats mostra que algunes seccions mal classificades apareixen de mitjana a 0.51 i 0.67 mm de distància de la classe que han observat els experts, pel lumen i media respectivament. Finalment, la mesura de Jaccard pel lumen és prou alta, de 0.85, mentre que pel media és de 0.60, que és bastant baixa. Els resultats trobats en aquest projecte presenten un bon començament sobre com implementar una solució pel problema explicat.

Resumen

La ecografía intravascular (IVUS) es una modalidad de imagen intraoperatoria que facilita la observación de la estructura de la pared del vaso de las arterias coronarias humanas. La segmentación de la pared arterial a partir de imágenes IVUS es crucial para el análisis de las características de la pared y para los modelos reconstruidos en 3D de la arteria. El objetivo de este proyecto es aprender, estudiar, desarrollar y evaluar un modelo capaz de realizar la segmentación semántica en imágenes médicas utilizando un pequeño conjunto de datos IVUS. El conjunto de datos utilizado contiene 435 extracciones de 10 pacientes diferentes adquiridas por sondas de 20 MHz. Nuestro modelo propuesto utiliza la arquitectura InceptionResNet preentrenada con el conjunto de datos ImageNet como codificador. A continuación, el modelo se conecta como una U-Net con un decodificador que se entrena utilizando las imágenes. Hemos explorado diferentes enfoques en el aumento de datos y los mejores resultados se obtuvieron con rotaciones de hasta 90 grados, giros verticales y horizontales y zoom. Entre los resultados, obtuvimos una media del coeficiente de similitud de Dice, o índice de Dice, de 0,75 para la sección media y de 0,92 para el lumen, con una varianza bastante baja. Sin embargo, la distancia de Hausdorff (HD) de los resultados muestra que algunas secciones mal clasificadas aparecen en promedio a 0,51 y 0,67 mm de distancia de la clase verdadera, para el lumen y la media respectivamente. Por último, la medida de Jaccard para el lumen es bastante alta: 0,85; mientras que para media es de 0,60, que es bastante baja. Los resultados encontrados en este proyecto presentan un buen comienzo sobre cómo implementar una solución para el problema explicado.

2021 Mathematics Subject Classification. 68T07

Contents

Chapter 1

Introduction

In this section we want to explain the motivation of this project, as well as, the planning of the time that we have dedicated to it.

1.1 Motivation and objectives

Cardiovascular diseases account for 30% of all deaths worldwide. The most common heart problems are coronary artery disease and heart failure, both of which are related to heart attacks. Coronary artery disease occurs when the arteries that supply blood to the heart muscle become narrowed or blocked by a buildup of plaque. Heart failure occurs when the heart is unable to pump enough blood to meet the body's demands.

Uncontrolled high blood pressure and high cholesterol can also lead to cardiovascular diseases. High blood pressure and high cholesterol can raise the risk of heart disease and stroke. Therefore, better detection and treatment of cardiovascular diseases are key in preventing their development and avoiding deaths. [34].

Intravascular ultrasound (IVUS) is the preferred modality for assessing lumen and vessel wall dimensions, quantifying plaque burden and guiding revascularisation in complex lesions and high-risk patients. However, the use of such modality has been limited by the time and cost needed for obtaining and analysing IVUS images.

In recent years, Deep Learning (DL) methodologies have found medical applications in providing fast and accurate processing of large datasets of images. These models rely on learning from large datasets of human annotations and use this information to train algorithmic models capable of replicating human performance within seconds. These characteristics probably make DL the ideal approach for IVUS images analysis. The aim of this project is to learn, study, develop and

validate a DL model capable of automatically segment IVUS data.

1.2 Project's planning

This project took about seven months to develop. The first meeting with Dr. Balocco was in July when we started choosing the topic of the project. At that time I was recommended to start learning about neural networks, how and why they work. After this, I started learning about the framework we were going to use to develop the project, **Keras**.

In September, we decided which problem we were going to face: the IVUS Challenge. From this point, we started working from the most basic neural network to more complicated ones, learning new things at each step. For the memory of this project, the idea was to write different reports as the project progressed. We can find a Gantt chart of the project's timeline in Fig. 1.1.

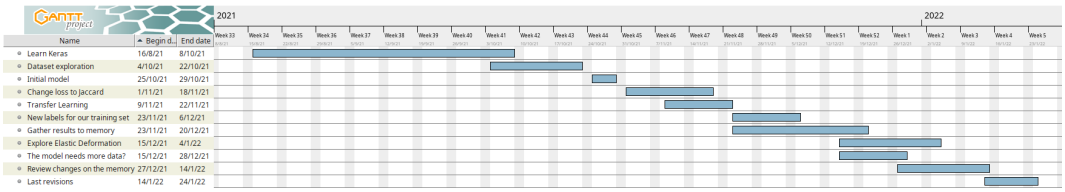


Figure 1.1: Gantt chart showing the project's timeline.

Chapter 2

State of the art

IVUS images semantic segmentation is a sensitive and quite challenging task. Thankfully, Balocco et al. [4] described an evaluation framework that allows a standardized and quantitative comparison between different IVUS lumen and media segmentation algorithms. Also, they describe the available IVUS dataset.

Over the years, many different automatic algorithms have emerged to try to solve this problem. One of the first is from 2008, when Downe et al. [15] used Principal Component Analysis to remove noise from IVUS images and other image processing techniques to detect the boundaries. Also, Faraji et al. [17] proposed a novel feature extraction method called Extremal Regions of Extremum Levels. However, these methods get clearly outperformed by DL methodologies. For example, Yang et al. [39] [38] proposed two U-Net-like architectures with two different data augmentations techniques which have very good and stable results.

New approaches have been investigated during this last year. For example, Bargsten et al. [6] optimized a capsule network architecture and concluded that it is a promising candidate when it comes to segmentation of small IVUS image datasets, achieving a Dice similarity coefficient of 0.94 for the lumen and 0.81 for the media and a Hausdorff Distance of 0.21 mm and 0.35 mm for the lumen and media respectively. We can see more results in Table 2.1.

Also, innovative and realistic data augmentation techniques seem to affect the results very positively for deep learning algorithms. Therefore, we will also have to focus on that part.

Table 2.1: Comparison of the performance of different IVUS images segmentation algorithms evaluated on the Volcano Dataset described at [4]. Their method and the data augmentation (if mentioned) are also described. The evaluation measures are Jaccard Measure (JM), Hausdorff Distance (HD) [mm] and Dice Index (DI). The average and standard deviation or an interval are provided.

Authors	Method	Data augmentation	JM_{lumen}	HD_{lumen}	DI_{lumen}	JM_{media}	HD_{media}	DI_{media}
Nandamuri et al. (2019) [28]	SUMNet: U-Net with the encoder similar to VGG11	-	0.95 ± 0.03	0.17 ± 0.07	-	0.97 ± 0.01	0.16 ± 0.09	-
Yang et al. (2018) [39]	IVUS-Net: fully convolutional network (FCN) with inspirations from aggregated, multi-branch architectures such as ResNeXT and the Inception model	Horizontal and vertical flips and Gaussian noise	0.90 (0.06)	0.26 (0.25)	-	0.86 (0.11)	0.48 (0.44)	-
Farahi et al. (2018) [17]	Extremal Regions of Extremum Levels (EREL)	-	0.91 (0.04)	0.22 (0.12)	-	0.83 (0.15)	0.50 (0.45)	-
Yang et al. (2019) [38]	DPU-Net	Masks designed to mimic shadow, side vessels and bifurcations	0.87 (0.07)	0.82 (0.61)	-	0.86 (0.09)	1.07 (0.72)	-
Balakrishna et al. (2018) [3]	U-Net with VGG16 as encoder	Horizontal and vertical flips, width and height shifts and rotations	0.7982 (-)	-	0.8846 (-)	0.8085 (-)	-	0.8825 (-)
Lo Vercio et al. (2019) [25]	Random Forest to detect structures and SVM for classification	-	0.83 (0.10)	0.57 (0.31)	-	0.88 (0.08)	0.32 (0.25)	-
Bargsten et al. (2021) [6]	Capsule networks	-	-	0.629 ± 0.123	93.52 ± 1.05	-	0.825 ± 0.116	79.11 ± 1.15
Downe et al. (2008) [15]	PCA for preprocessing, active contours and graph search for segmentation	-	0.77 (0.09)	0.47 (0.22)	-	0.74 (0.17)	0.76 (0.48)	-

Chapter 3

Neural networks: deep learning

3.1 Definitions

Definition 3.1. *An **Artifical Neural Network (ANN)** or simply a **Neural Network (NN)** is a computing system designed to simulate the functions of the neural network of the human brain. An ANN is based on a collection of connected units called **artificial neurons**.*

Definition 3.2. *An **artificial neuron** is a mathematical function conceived as a model of biological neurons, a neural network. The artificial neuron receives one or more inputs and sums them to produce an output. Usually each input is separately weighted, and the sum is passed through a non-linear function known as an activation function or transfer function.*

Definition 3.3. *A neural network **layer** is a collection of neurons operating together at a specific depth within a neural network. We can distinguish three different types of layers: the input layer, which is the layer that receives external data; the hidden layers, layers between the input and the output layers; and the output layer, which is the layer that produces the ultimate result.*

Definition 3.4. ***Learning** is the adaptation of the network to better handle a task by considering sample observations. This process involves adjusting weights of the network to improve the accuracy of the result.*

Definition 3.5. *A neural network **parameter** is a coefficient of the model, which is chosen by the model itself. These parameters can be slightly modified in order to make the model improve its performance.*

Definition 3.6. *A neural network **hyperparameter** is a constant parameter whose value is set before the learning process begins.*

Definition 3.7. *A **loss function** or **cost function** is a function that maps an event or values of one or more variables onto a real number. Intuitively, the output represents some "cost" associated with the event. An optimization problem seeks to minimize a loss function.*

Definition 3.8. *A **deep neural network** is a neural network with multiple hidden layers.*

3.2 Working environment

In this section we will present the software and hardware technologies used throughout this final project.

3.2.1 Programming language and libraries

In this work we will be using Python as our programming language. Python is an interpreted high-level general-purpose programming language. It is meant to be an easily readable language. The language core philosophy is summarized in the document *The Zen of Python (PEP 20)*:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readibility counts.

Interpreted programming languages cannot be executed as fast as compiled languages such as C. In Python, we can leverage libraries written in Python and C, such as **NumPy**. NumPy is an open-source project that enables numerical computing in Python. It also works as the core for other libraries, such as TensorFlow, SciPy, Matplotlib, Pandas, etc.

The environment we will be working with is **Jupyter Notebook**. Jupyter Notebook is an open-source web application used to create interactive documents that contain live code, equations, visualizations and narrative text. These documents are divided into cells which can be compiled into text and visualizations or executed as Python code.

3.2.2 Keras and TensorFlow 2

Keras is an open source Deep Learning API written in Python. It was developed to enable fast experimentation [23]. Keras runs on top of **TensorFlow**, an end-to-end open source platform for machine learning. We can think of TensorFlow as an infrastructure layer for *differentiable programming*. The most notable abilities of TensorFlow are:

- Efficiently executing tensor operations on CPU, GPU or TPU.
- Computing the gradient of arbitrary differentiable expressions.
- Scaling computation to many devices.
- Exporting programs to external runtimes such as servers, browsers, mobiles and embedded devices.

Keras, as the high-level API of TensorFlow 2, is a highly productive interface for solving Maching Learning problems, with a focus on Deep Learning. It is a tool that provides the ability to build blocks for developing Machine Learning solutions with high iteration velocity.

3.2.3 Hardware

Since the problem we will be working on requires low computational power, there is no need to use any external hardware besides a personal computer with a dedicated GPU. In this case, we will be using the following specifications:

- Operating System: Ubuntu 20.04
- CPU: Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz
- RAM: 16GB
- GPU: NVIDIA GeForce GTX 1060 with Max-Q Design

3.3 How do neural networks work?

ANNs simulate the human nervous system and can learn and generalize from examples to produce meaningful solutions even when the input data contains errors or is incomplete.

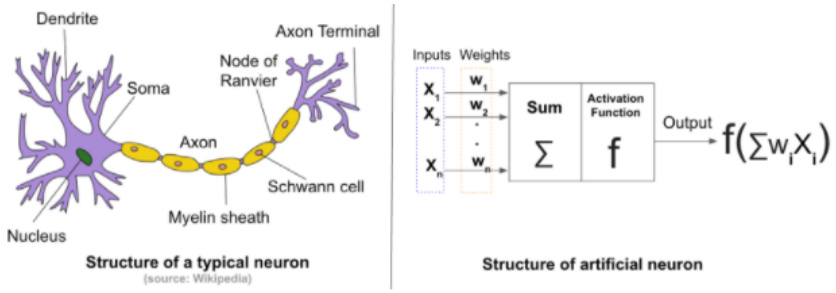


Figure 3.1: Structure of a brain neuron vs ANN neuron

3.3.1 Similarities with the human brain

A human brain has a biological network of billions of interconnections between neurons that can change, grow or be removed as the brain learns. The structure of a neuron can be seen in Fig. 3.1. There is a dendrite that takes input from other neurons and feeds it to the nucleus. The signals received are accumulated in the nucleus and if it exceeds a threshold, the axon sends a signal down to the other neurons. The neuron is either **activated** or **inhibited** based on the level of accumulation. Now, if we compare it with the artificial neuron, we clearly see that they take input from the output of other neurons, they accumulate this output and apply an activation function similar to the threshold that biological neurons have. However, the way they take input in each case is different. In our understanding of biological neural networks, we know that the input is taken from dendrites and output through the axon and these have different ways. Research shows that dendrites themselves apply a non-linear function on the input before it is passed to the nucleus.

So we can think of an artificial neuron as a its own linear regression model composed of input data, weights, a bias (or threshold) and an output, but with an activation function. Therefore, an artificial neural network is a mathematical model that we can interpret as a directed graph, as we can see in Fig. 3.2. The computation of an ANN begins with an input array of numbers x_i that is taken by the input layer. Then these signals move along connections to each of the nodes in the adjacent layer and can be inhibited or amplified through connection weights, w_t. The adjacent layer's nodes are either activated or inhibited depending on the incoming signals and they output another signal, O_j, by passing them through an activation function. For example, one of the most famous activation functions is the **logistic** or **sigmoid function**:

$$\sigma : \mathbb{R} \longrightarrow (0,1)$$
$$x \longmapsto \frac{1}{1+e^{-x}}$$

Hence the output of the neuron is calculated as:

$$O_j = \frac{1}{1+e^{-\sum x_i w_i}}.$$

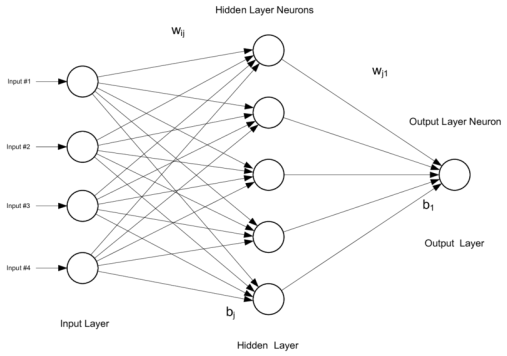


Figure 3.2: Structure of a typical artificial neural network

Now, let's have a look at the most common **activation functions**.

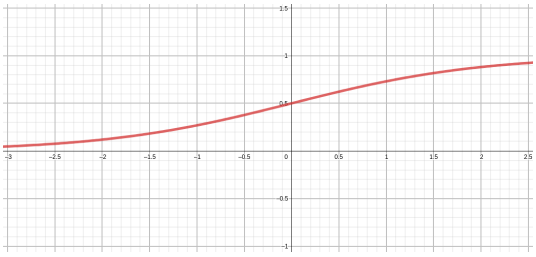
3.3.2 Activation functions

- **Binary step function**, or Heaviside step function, usually denoted by H or θ is a threshold-based activation function. In this case, the neuron can only send two different values. This is the most simple activation function and is usually used with threshold 0, so it is defined as

$$H(x) := \mathbb{1}_{x>0} = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

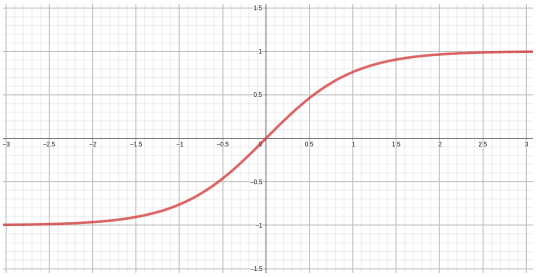
- **Logistic function**, or sigmoid function, is another important activation function, it has a smooth gradient and makes predictions very clear, since values below -2 and above 2 are mapped very close to 0 and 1 respectively. It is defined as follows

$$f : \mathbb{R} \longrightarrow (0,1)$$
$$x \longmapsto \frac{1}{1+e^{-x}}$$



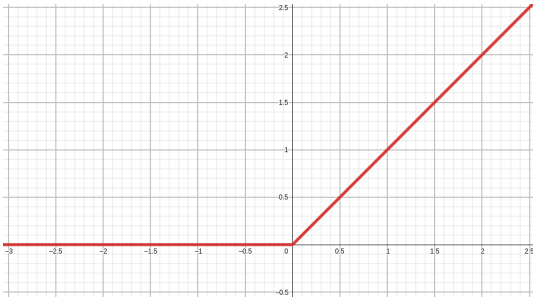
- **Hyperbolic tangent** is a very well-known function and is quite similar to the sigmoid function. However, its range is $(-1,1)$, it is centered at 0 and its slope near 0 is much higher than the sigmoid's. It is defined as follows

$$\tanh : \mathbb{R} \longrightarrow (-1,1)$$
$$x \longmapsto \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$



- **Rectified linear unit (ReLU)** is probably the most used activation function in deep learning nowadays. It retains the properties of linear models which makes it better for gradient propagation. Even though it is not differentiable at zero, the value of the derivative at zero can be arbitrarily chosen to be 0 or 1. The function is defined as

$$f(x) := \max(0,x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$



3.4 How do neural networks learn?

There are two main phases in the life cycle of a neural network: the training phase and the prediction phase. Here, we will discuss the former. Since this project aims to create a neural network via images and its labels, we will talk about the training phase in **supervised learning**.

The process of finding the optimal weights and biases values is what we call **learning**, is actually the **training phase**. In order to find optimal values for weights and values, neural networks use an **error gradient**. Using this method, we want to know, when fixing values for weights and biases, whether these values are bigger or smaller with respect to their optimal value, and thus whether they need to be increased or decreased, and how much. Therefore, the gradient we are looking for is the partial derivatives of the error or loss function with respect to the weights and biases: $\frac{\partial E}{\partial w}$, $\frac{\partial E}{\partial b}$, where E is the error or loss function, w is weight and b bias.

3.4.1 Loss functions

The error of an output with respect to its desired value is calculated using a loss function L. Therefore, the main goal of the training phase is to minimize this function.

Now, we will see some examples of loss functions:

- **Mean Squared Error (MSE)**.

$$L_y(\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

  where $\hat{y}$ is the output vector and y is the vector of expected values.

- **Binary Crossentropy**. It is usually used when there is a classification over two different categories, when the target labels are usually 1 and 0.

$$L_y(\hat{y}) = -y \cdot \log(\hat{y}) - (1-y) \cdot \log(1-\hat{y})$$

  So when the predicted label is close to 1, $\log(\hat{y}_i)$ becomes close to 0 and $\log(1-\hat{y}_i)$ becomes close to $-\infty$, so if the actual label is 1, the loss function is nearly 0, and if it is 0, the loss function is a very large value. Analogous for $\hat{y}_i \approx 0$.

- **Categorical Crossentropy**. It is a generalization of the binary crossentropy loss function to C possible categories.

$$L_y(\hat{y}) = \sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

  Usually, in this case, $\hat{y}$ is passed through an activation function called **softmax** that normalizes the input into C probabilities proportional to the exponential of the input, so we have $\sum_{i=1}^{C} \hat{y}_i = 1$.

- **Jaccard loss**. It is a loss function created from the Jaccard similarity coefficient, also known as Intersection over Union (IoU). It is a statistic used to determine the similarity of two areas that overlap.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

So the Jaccard loss is defined as follows

$$L(A, B) = 1 - J(A, B)$$

3.4.2 Backpropagation

In order to compute the error gradient, we use a very well-known algorithm called backpropagation. Firstly, the error, or loss, is computed from the output of the neural network. Then, the derivatives of the error with respect to each layer's weights and biases $\frac{\partial L}{\partial w_n}$ $\frac{\partial L}{\partial b_n}$ are computed using the chain rule with its following layer. So we have,

$$\begin{cases} \frac{\partial L}{\partial w_n} = \frac{\partial L}{\partial a_n}\frac{\partial a_n}{\partial w_n} \\ \frac{\partial L}{\partial b_n} = \frac{\partial L}{\partial a_n}\frac{\partial a_n}{\partial b_n} \end{cases}, \text{ where } a_n \text{ is the activation function of the } n\text{-th layer.}$$

At this point, we iterate backwards and compute the derivatives of the previous layers' weights and biases, of the i-th layer, with respect its next layer, the $(i+1)$-th layer, and use the chain rule to compute the error gradient. This is why it's called *backpropagation*.

Once the gradient has been computed, we need to improve our weights and biases. The way the error gradient values are used is through an optimization algorithm. One simple but frequently used example of such algorithms is **gradient descent**.

3.4.3 Gradient descent

Gradient descent is a first-order iterative optimization algorithm, because at most it only takes into account the local error. It is used for finding a local minimum of a differentiable function, in this case the loss function L. The idea is to take steps in the opposite direction of the gradient at each iteration, because this is the direction of the steepest descent. The algorithm is defined as follows:

$$(w_i)_{n+1} = (w_i)_n - \alpha \frac{\partial L}{\partial (w_i)_n}$$

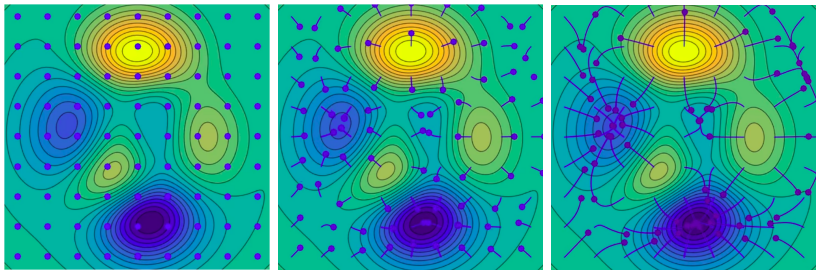


Figure 3.3: Gradient descent in 2D where the blue sections represent local minimums.

where $(w_i)_n$ is the value of the weight w_i at step n, and $\alpha > 0$. If α is small enough, then $L_n > L_{n+1}$, where L_n is the value of the loss function L at step n. We can see an illustrated example in Fig. 3.3.

This method requires the whole dataset to be computed through the network and backpropagated at every iteration. Therefore, it does not scale very well with big datasets. That is why different algorithms based on gradient descent have been developed over the last few years, like batch gradient descent, stochastic gradient descent or mini-batch gradient descent [29], they mainly differ in the amount of data they use. Also, there exist other gradient descent algorithms that outperform the gradient descent in terms of convergence, such as Momentum, AdaGrad, RMSProp or Adam [18]. This last one has empirically shown, in recent years, that in most cases, it works better than the rest, since it takes the best of Momentum and RMSProp algorithms and uses it.

3.5 Deep learning

Deep learning is part of a family of machine learning methods based on artificial neural networks. Deep learning consists on using deep neural networks. As we defined before, a deep neural network (DNN) is an artificial neural network (ANN) with multiple hidden layers.

DNNs can model complex nonlinear relationships. In order to do so, they need to have nonlinear activation functions and other complex relations between layers.

3.5.1 The need for nonlinear activation functions

In 1989, Hornik, Stinchcombe and White published an article proving that for any continuous function f on a compact set K, there exists a feedforward neural network, with only a single hidden layer, which uniformly approximates f to within an arbitrary $\epsilon > 0$ on K, this is called **The Universal Approximation**

Theorem for Neural Networks [24]. In this section we will try to present the same facts in a more concise way and to explain the intuition behind it.

Firstly, we denote the n-dimensional unit cube by I_n, which can be represented as the cartesian product $[0,1]^n$. Additionally, we denote $C(I_n)$, the space of continuous functions, with codomain $\mathbb{C}$, on I_n. Also, we define $M(I_n)$ to be the space of finite, signed regular Borel measures on I_n.

Intuitively, a **regular** measure on a topological space is a measure for which every measurable set can be approximated from above by open measurable sets and from below by compact measurable sets. From now on, all the integrations done will be with respect to regular measures on $C(I_n)$.

Since we are working in a metric space, we say that A is **dense** in X, if for every $x \in X$, there exists a sequence $\{a_n\}_{n\in\mathbb{N}}$ such that $\lim_{n\to\infty} a_n = x$. Now, we can reformulate the uniform approximation theorem in the following different ways:

1. The set of all feedforward neural networks $\mathcal{N}$ is dense in $C(I_n)$.
2. For every continuous function $f \in C(I_n)$, there exists a sequence of neural networks $\{n_i\} \in \mathcal{N}$ converging to f, that is $\lim_{i\to\infty} n_i = f$.
3. For every continuous function $f \in C(I_n)$ and $\epsilon > 0$ there exists a neural network $g \in \mathcal{N}$ such that $||g - f|| < \epsilon$.

So, in essence, what we are trying to prove is that $\widetilde{\mathcal{N}} = C(I_n)$. Before proceeding to the theorem, we will first define some concepts.

Definition 3.9. *A **feedforward neural network** having N neurons arranged in a single hidden layer is a function $y : \mathbb{R}^d \longrightarrow \mathbb{R}$, $d > 0$, of the form*

$$y(x) = \sum_{i=1}^{N} \alpha_i \sigma(\mathbf{w}_i^T \mathbf{x} + b_i)\,,$$

*where $\mathbf{w}_i, \mathbf{x} \in \mathbb{R}^d$, $\alpha_i, b_i \in \mathbb{R}$, and σ is a nonlinear **sigmoidal** activation function.*

As we have seen before, $\mathbf{w}_i$ is the array of weights that are applied to the input $\mathbf{x}$. The α_i are the network weights applied to the output of each unit in the hidden layer. Finally, b_i is the bias of unit i. Therefore, a neural network of this type is just a linear combination of affine transformations under a sigmoidal activation function.

Definition 3.10. *A **sigmoidal activation function** is a function $\sigma : \mathbb{R} \longrightarrow \mathbb{R}$ that satisfies*

$$\sigma(t) \to \begin{cases} 1 \text{ as } t \to \infty \\ 0 \text{ as } t \to -\infty \end{cases} .$$

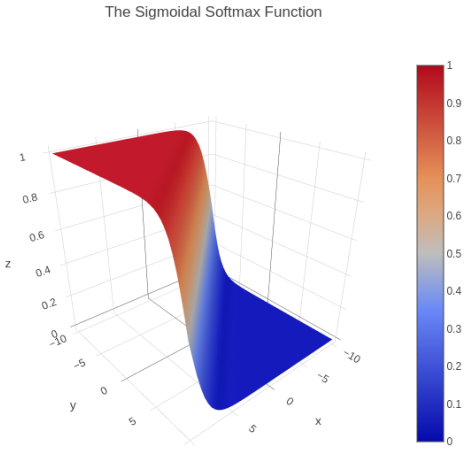


Figure 3.4: The Sigmoidal Softmax Function

A very famous sigmoidal function is the **softmax function**, which we can see in Fig. 3.4.

Definition 3.11. *We say that a sigmoidal function σ is **discriminatory** if for every $\mu \in M(I_n)$ and*

$$\int_{I_n} \sigma(\mathbf{w}_i^T \mathbf{x} + b_i) d\mu(\mathbf{x}) = 0, \quad \forall \mathbf{w}_i \in \mathbb{R}^d, b_i \in \mathbb{R}$$

then $\mu = 0$.

This last definition tells us that a discriminatory sigmoidal activation function σ, can't send the affine space $\mathbf{w}_i \mathbf{x} + b_i$ to a set of measure zero. So, in some sense, these limit properties are preventing the function from *losing* information from the input.

Now, before diving into the proof of the theorem, let's announce first a couple of famous theorems that will be useful. The proofs of both theorems are beyond the scope of this final project.

Hahn-Banach Theorem. Let X be a real vector space and p a sublinear functional on X. Let f be a linear functional defined on a subspace $Y \subset X$, satisfying $f(y) \leq p(y) \, \forall y \in Y$. Then f has a linear extension $\widetilde{f}$ from Y to X satisfying

(a) $\widetilde{f}$ is a linear functional on X,

(b) $\widetilde{f}|_Y = f$, i.e. the restriction of $\widetilde{f}$ to Y agrees with f,

(c) $\widetilde{f}(x) \leq p(x) \quad \forall x \in X$.

Riesz Representation Theorem. Let $C_c(X)$ denote the functions of compact support defined on X. If I is a positive linear functional on $C_c(X)$, there is a unique Radon measure μ on X such that $I(f) = \int f\, d\mu, \quad \forall f \in C_C(X)$. Moreover, μ satisfies

$$\mu(U) = \sup\{I(f) : f \in C_c(X),\, 0 \leq f \leq 1,\, \text{supp}(f) \subset U\}\,, \quad \forall U \subset X \text{ open}$$

where $\text{supp}(f) = \{x \in X : f(x) \neq 0\}$, and

$$\mu(K) = \inf\{I(f) : f \in C_c(X),\, 0 \leq f \leq 1, f \geq \mathbb{1}_K\}\,, \quad \forall K \subset X \text{ compact}$$

where $\mathbb{1}_K$ is the indicator function of the compact set K.

Theorem 3.12. *If the activation function σ is a continuous, discriminatory function, then the set of all feedforward neural networks $\mathcal{N}$ is dense in $C(I_n)$.*

Proof. Let $\mathcal{N} \subset C(I_n)$ be the set of neural networks. To prove that $\mathcal{N}$ is dense in $C(I_n)$, we will prove that its closure $\widetilde{\mathcal{N}}$ is $C(I_n)$, and we will do so by way of contradiction. Suppose that $\widetilde{\mathcal{N}} \neq C(I_n)$, then $\widetilde{\mathcal{N}}$ is a closed, proper subspace of $C(I_n)$. By the Hahn-Banach Theorem, there exists a bounded linear functional on $C(I_n)$, let's call it L, such that $L(h) = 0$ for every $h \in \widetilde{\mathcal{N}}$ but $L \neq 0$. Now, since $C(I_n)$ is σ-compact, we do not have to deal with Radon measures and we can restrict ourselves to the regular Borel measures. By the Riesz Representation Theorem, we can write the functional L in the following form

$$L(h) = \int_{I_n} h(x) d\mu(x), \quad \text{for some } \mu \in M(I_n), \text{ for all } h \in C(I_n)$$

In particular, since by definition any feedforward neural network is a member of $\mathcal{N}$, and L is identically zero on $\mathcal{N}$, we have

$$L|_{\mathcal{N}}(h) = \int_{I_n} h(x) d\mu(x) = 0$$

And because we took σ to be discriminatory, we must have $\mu = 0$. But this contradicts our determination that $L \neq 0$, because

$$\mu = 0 \implies \int_{I_n} h(x) d\mu(x) = 0 \quad \text{for all } h \in C(I_n) \implies L(h) = 0 \quad \text{for all } h \in C(I_n)$$

Hence $\mathcal{N}$ is dense in $C(I_n)$. □

So, we have proved what we wanted to see. Let us remark, that there also exists a variety of other universal approximation theorems between non-Euclidean spaces and other commonly used architectures, such as the CNN architecture.

Nonetheless, we have not shown that there exists a **discriminatory** sigmoid function $\sigma(t)$. Let's do that.

Lemma 3.13. *Any bounded, measurable sigmoidal function σ is discriminatory. In particular, any continuous sigmoidal function is discriminatory.*

Proof. For any $x, y \in \mathbb{R}^d$, $d > 0$, $\theta, \varphi \in \mathbb{R}$ we have

$$\sigma_\lambda(x) = \sigma(\lambda(y^T x + \theta) + \varphi) = \begin{cases} \to 1 & \text{when } y^T x + \theta > 0 \text{ as } \lambda \to \infty \\ \to 0 & \text{when } y^T x + \theta < 0 \text{ as } \lambda \to \infty \\ = \sigma(\varphi) & \text{when } y^T x + \theta = 0 \end{cases} \quad \lambda > 0\,.$$

Thus, the function $\sigma_\lambda(x)$ converges pointwise and boundedly to

$$\gamma(x) = \begin{cases} 1 & \text{when } y^T x + \theta > 0 \\ 0 & \text{when } y^T x + \theta < 0 \\ \sigma(\varphi) & \text{when } y^T x + \theta = 0\,, \end{cases}$$

as $\lambda \to \infty$. Let $\Pi_{y,\theta} = \{x \in \mathbb{R}^d | y^T x + \theta = 0\}$ be an affine hyperplane and let $H_{y,\theta}$ be the open half-space defined by $\{y^T x + \theta > 0\}$. Note that $|\sigma_{\lambda(x)}| \leq \max(1, \sigma(\varphi))$ for all $x \in \mathbb{R}^d$. Therefore, we can apply the Dominated Convergence Theorem to swap the limit and the integral to get

$$\lim_{\lambda\to\infty} \int_{I_n} \sigma_{\lambda(x)}\, d\mu_x = \int_{I_n} \lim_{\lambda\to\infty} \sigma_{\lambda(x)}\, d\mu_x = \int_{I_n} \gamma(x)\, d\mu(x) = \sigma(\varphi)\mu(\Pi_{y,\theta}) + \mu(H_{y,\theta})$$

for all $y \in \mathbb{R}^d$, $\varphi, \theta \in \mathbb{R}$.

We want to show that if the above integral is equal to zero, then $\mu = 0$. Analogously, we must show that the measure of all half-planes being 0 implies that the measure μ equals 0. To do this, we fix y and, for a bounded, measurable function h, we define a linear functional F as follows

$$F(h) = \int_{I_n} h(y^T x)\, d\mu(x)\,.$$

Note that $F \in L^\infty(\mathbb{R})$, i.e. F is a bounded linear functional. This is because when we integrate with respect to a finite measure, we cannot get an infinite result. Now, let $h := \mathbb{1}_{[\theta,\infty)}$ be the indicator function for the interval $[\theta, \infty)$ so that

$$F(h) = \int_{I_n} h(y^T x)\, d\mu(x) = \mu(\Pi_{y,-\theta}) + \mu(H_{y,-\theta}) = 0\,.$$

Thus, if $y^T x \in [\theta, \infty)$, then $y^T x - \theta \geq 0$. For the integral, this decomposes into the measure of two disjoint sets, the hyperplane $\Pi_{y,-\theta} = \{x \in \mathbb{R}^d : y^T x - \theta = 0\}$ and the half-space $H_{y,-\theta} = \{x \in \mathbb{R}^d : y^T x - \theta > 0\}$. Similarly, $F(h) = 0$ if $h = \mathbb{1}_{(\theta,\infty)}$ is the indicator function for the open interval (θ, ∞). By linearity, F is 0 for the

indicator function on any interval and hence for any simple function, due to the fact that simple functions are dense in $L^\infty(\mathbb{R})$, so $F = 0$.

In particular, we can take the bounded, measurable functions $s(u) = \sin(m \cdot u)$ and $c(u) = \cos(m \cdot u)$, and it gives

$$F(s + ic) = \int_{I_n} \cos(m^T x) + i \sin(m^T x)\, d\mu(x) = \int_{I_n} e^{im^T x}\, d\mu(x) = 0, \quad \forall m\,.$$

Hence, the Fourier transform of μ is 0 and so μ must be null as well. Therefore σ is discriminatory. □

So now we wonder why is this such an important result. Firstly, note that a linear neural network, i. e. a feedforward neural network with linear activation functions and n layers each having m hidden units, is equivalent to a linear neural network without hidden layers. The proof is very simple:

$$\begin{aligned} y &= h(\mathbf{x}) \\ &= b_n + w_n(b_{n-1} + w_{n-1}(\dots(b_1 + w_1\mathbf{x})\dots)) \\ &= b_n + w_n b_{n-1} + w_n w_{n-1} b_{n-2} + \dots + w_n \dots w_1 \mathbf{x} \\ &= b' + w'\mathbf{x} \end{aligned}$$

Thus it's clear that adding layers does not increase the approximation power of a linear neural network at all. So we need nonlinear neural networks.

Since we will deal with images, we will now take a look at how deep learning is used in computer vision.

3.5.2 Computer vision

Deep learning is used pretty often in computer vision for highly complex problems such as image classification and image segmentation.

Image classification is the process of taking an **input** (a picture) and outputting a **class** or a **probability** that the input belongs to a particular class.

So, how can a computer learn to do that? The answer is with a **CNN** (convolutional neural network). A CNN is fast and efficient and is used widely from photo tagging [30] to self-driving cars [14] and also in healthcare or security. A CNN normally uses these layers:

- **Convolutional layers**:

  The main purpose of the convolutional layers is to extract features from the input image. In order to accomplish this, the convolution step applies a filter to each possible position of the image, this is done by multiplying the values

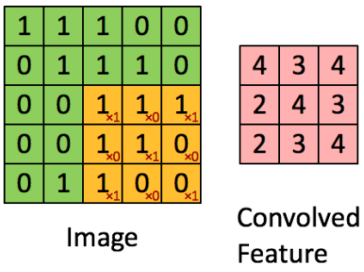


Figure 3.5: Convolution step. It multiplies each number in each 3×3 submatrix by the numbers represented in red as subindices and outputs the sum.

of a matrix by the values of the image. A representation can be seen in Fig. 3.5.

The result after the convolution step is called a **feature map** or an **activation map**. Using more the one filter will do a better job of preserving spatial relationships.

- **ReLU layers**:

  The ReLU (rectified linear unit) layer is a step that applies an activation function onto the feature map in order to increase nonlinearity in the network. It basically sets all negative values to zero.

- **Pooling layers**:

  Most of the time we find images that are flipped, rotated, and so on. Therefore, no matter what the size or the location of a feature in an image, we want it to be detected by the CNN. For this, we use pooling.

  Pooling reduces the size of the input representation and also makes it possible to detect objects in an image no matter where it is located. Moreover, this helps control overfitting, since it helps the CNN to understand the information instead of memorizing it.

  The most common example of pooling is **max pooling**. In max pooling, the image is separated into different areas that don't overlap with each other and the output is the maximum value of each area. The result of pooling a feature map is called a **pooled feature map**.

- **Fully connected layers**:

  Finally, fully connected layers are used after all the feature extraction processes. Artificial neural networks are really good at combining features into attributes and predicting classes with great accuracy.

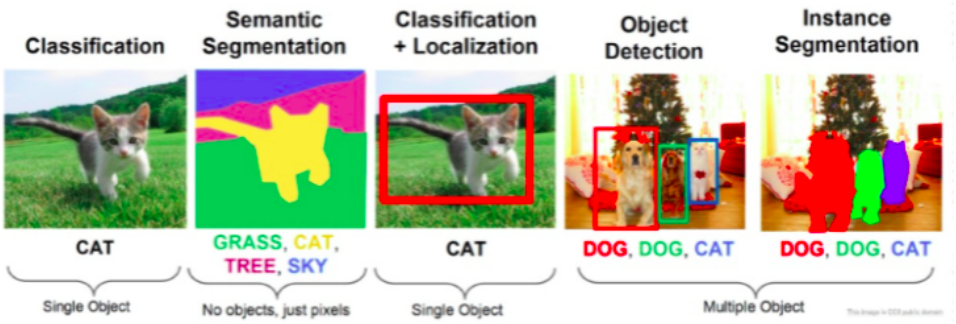


Figure 3.6: Different computer vision tasks. Source: Introduction to Artificial Intelligence and Computer Vision Revolution (`https://www.slideshare.net/darian_f/introduction-to-the-artificial-intelligence-and-computer-vision-revolution`).

A classic CNN architecture would look something like this:
Input → Convolution → ReLU → Convolution → ReLU → Pooling → ReLU → Convolution → ReLU → Pooling → Fully Connected

Image segmentation, also known as semantic segmentation, is the process of dividing a digital image into multiple subgroups called segments. The goal of segmentation is to simplify or change the representation of an image into something easier to analyze. In other words, segmentation consists of assigning a label to each pixel so that all pixels belonging to the same category have a common label. Image segmentation has many applications in medical imaging, self-driving cars and satellite imaging to name a few.

There are two classes of segmentation techniques:

- Classical computer vision approaches
- AI based approaches

In this project we will use the latter.

3.5.3 Transfer learning

Transfer learning in a machine learning process that focuses on taking advantage of the knowledge gained on one problem and apply it on a different but related one. For example, a model that has learned to recognize cars in images can be used as a starting point for a model to recognize trucks. Although it might seem very limited, from the practical point of view, this can mean improving resources efficiency, such as computational time or data samples used in the training phase.

The proper definition of transfer learning is given in terms of domains and tasks.

Definition 3.14. *A domain $\mathcal{D}$ is a tuple $(\mathcal{X}, \mathcal{P}(X))$, where $\mathcal{X}$ is a feature space and $\mathcal{P}(X)$ is a marginal probability distribution where $X = \{x_1, \ldots, x_n\} \in \mathcal{X}$.*

Definition 3.15. *Given a domain $\mathcal{D} = (\mathcal{X}, \mathcal{P}(X))$, a task $\mathcal{T}$ is a tuple $(\mathcal{Y}, f(x))$, where $\mathcal{Y}$ is a label space and $f : \mathcal{X} \longrightarrow \mathcal{Y}$ is an objective predictive function. The function f is used to predict the corresponding label $f(x)$ of a new instance x. A task $\mathcal{T}$ is learned from the training data consisting of pairs (x_i, y_i), where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.*

Given a source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and a learning task $\mathcal{T}_T$, where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$.

In computer vision, transfer learning is commonly used for object detection and object localization.

3.5.4 Network types

ImageNet is a public image database in which we can find about 14 million hand-labelled annotated images containing over 22 thousand day-to-day categories. Every year, an ImageNet competition is hosted in which a smaller version of this dataset is used with the aim of classifying the images. Therefore, new state of the art CNN architectures have emerged trying to beat the best possible accuracy thresholds. We will discuss some of the best solutions: VGG16, InceptionNet, ResNet and DeepLab.

- **VGG16** was published in 2014 and is the simplest architecture among others used in ImageNet competition. The VGG16 architecture contains a total of 16 layers in which weights and biases are trainable. Out of these, 13 are convolutional layers and are stacked one after the other, followed by 3 dense layers at the end for classification. A representation can be seen in Fig. 3.7.

  Its key features are:

  1. The number of filters in the convolution layers follow an increasing pattern similar to a decoder architecture.
  2. The most valuable features are obtained by max pooling layers at different steps in the architecture.

  One of the cons of this architecture is that it is slow to train and the final model that it produces is very large.

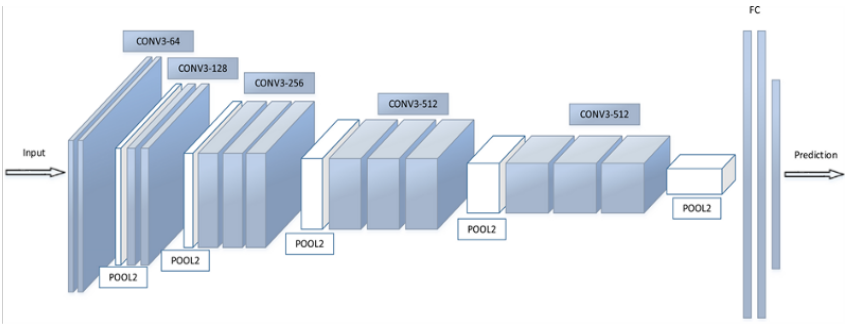


Figure 3.7: VGG16 architecture

- **InceptionNet**, also known as GoogleNet, produced the winning model of the 2014 ImageNet competition. It consists of 22 layers in total. The fundamental blocks of InceptionNet are called Inception modules, these are shown in Fig. 3.8. Its key features are:

  1. The Inception modules, or blocks, act as a multi-level feature extractor in which convolutions of different sizes are obtained to create a diversified feature map.
  2. They contain 1×1 convolution blocks that perform dimensionality reduction. By doing so, the inception block preserves the spatial dimensions but reduces the depth. So that the overall network's dimensions are not increased exponentially.

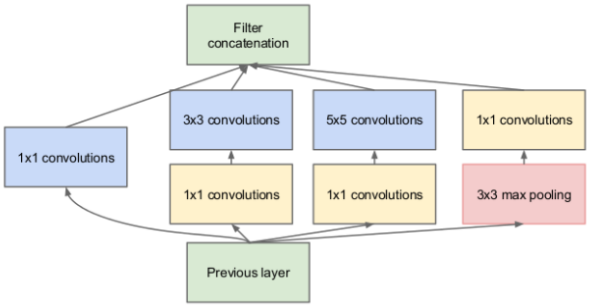


Figure 3.8: Inception module

The complete architecture can be found in Fig. 3.9.

- **ResNets**, also known as residual networks, try to overcome some common problems of very deep networks. All the previous models used deep neural networks in which they stacked many convolutional layers one after the

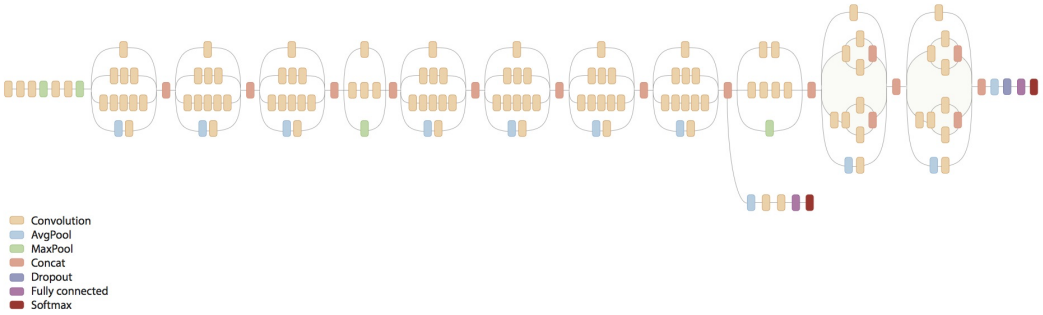


Figure 3.9: InceptionNet complete architecture

other, thinking that the deeper the network the better it performs. However, this turns out to be not completely true, deeper networks come with some problems:

- Networks become difficult to optimize due to vanishing/exploding gradients. Vanishing gradients happen, for example, when the sigmoid function is used as an activation function, its derivative becomes smaller as values become larger. If we happen to stack a lot of layers with this activation function, the gradient of the weights and biases in the beginning of the network become insignificant.
- Degradation problem, the accuracy gets saturated and then degrades quickly.

To address these problems, the ResNet architecture comes with skip connections as we can see in Fig. 3.10.

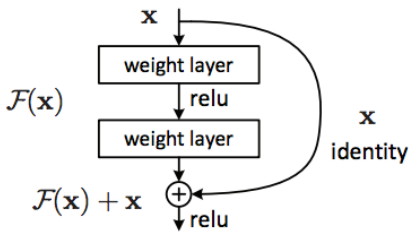


Figure 3.10: Skip connection architecture inside ResNet

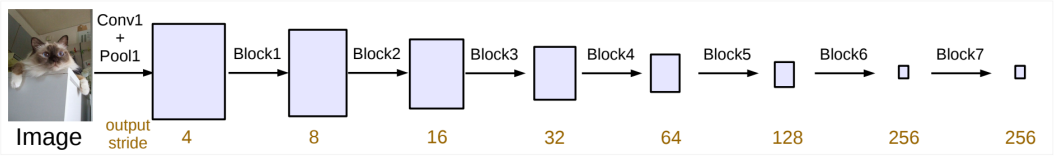
By doing this, we add the output of the activation function from shallower layers to the activation function their next layers. In order to do this, one

must ensure that the same dimensions of convolutions throughout the networks are used, for example 3×3 convolutions.

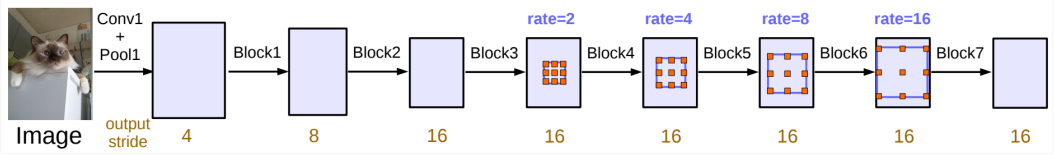
By using residual blocks in the network, we can build networks of any depth with the assumption that new layers are helping underlying layers to learn from the input data. Kaiming He, X. Zhang, Shaoqing Ren and Jian Sun [19] were able to create very deep neural networks of up to 152 layers with very high accuracy in ImageNet competitions.

- **DeepLab** is a semantic segmentation architecture. First, the input image goes through the network with the use of dilated convolutions. Then the output from the network is bilinearly interpolated and goes through the fully connected Conditional Random Fields [10] to fine tune the result and we obtain the final predictions.

  Newer versions of this architecture have been developed: DeepLabv2 and DeepLabv3. **DeepLabv3** is a semantic segmentation architecture that improves upon DeepLabv2 with several modifications. To handle the problem of segmenting objects at multiple scales, modules are designed so that they employ atrous convolution in cascade or in parallel to capture multi-scale context by adopting multiple atrous rates (Fig. 3.11). Furthermore, the Atrous Spatial Pyramid Pooling module from DeepLabv2 augmented with image-level features encoding global context and further boost performance [9]. We can see an illustrative representation in Fig. 3.12.



(a) Going deeper without atrous convolution.



(b) Going deeper with atrous convolution. Atrous convolution with $rate > 1$ is applied after block3 when $output\_stride = 16$.

Figure 3.11: Cascaded modules without and with atrous convolution.

Also, there exists a hybrid module called Inception-Resnet inspired by both Inception and the performance of ResNet architecture, which uses the inception modules with residual connections to accelerate and improve its learning process significantly. There exists version 1 and version 2 of this architecture. The latter has higher computational cost and accuracy.

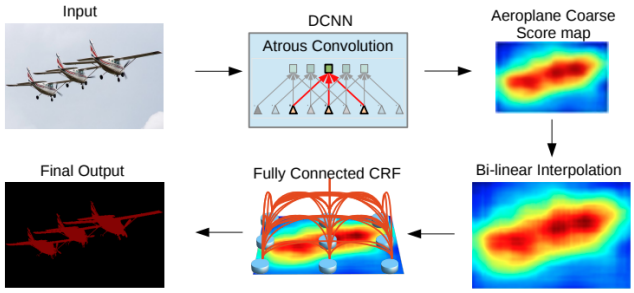


Figure 3.12: Process to perform semantic segmentation using atrous convolution and CRFs.

3.5.5 Creating models in Keras

Layers are the basic building blocks of neural networks in Keras. The easiest way to build a model is by using the `Sequential` class. As its name suggests, it helps us concatenate several layers to build a model. Moreover, we can also get a summary of its parameters in each layer. We can see an example in A.1.

The model on A.1 is used to detect handwritten digits, from 0 to 9, in 28×28 black and white images. Therefore, we have `input_shape=(28, 28, 1)`. The layers used are basically 2D convolution layer with **Rectified Linear Unit** (ReLU) activation function followed by a **max pooling** operation for 2D data. This combination of layers helps the model produce nonlinear filters for every 3×3 window and take the most significant part out of it. At the end, we create a neural network between all the nodes that we got after two convolutional layers and ten new nodes which represent each class (digits from 0 to 9).

In case we do not want to create a sequential model, but a more complex one, we can define a model by using the output from a layer as the input of other layers. We can see an example of this for image segmentation in A.1.

The example is based on the famous architecture U-Net convolutional neural network. The model is divided in two halves. The first half is a contracting path, called encoder, which follows the typical convolutional network. During this process, the space of the information is reduced, while the feature information is increased. After that, in the second half which is an expansive path, also called decoder, the space of the information is increased through a combination of up-convolutions, preserving the feature information obtained in the contracting path.

The contracting and expansive path give the model a u-shaped architecture, we can see an illustration of this model in Fig. 3.13.

However, we can see that our model does not use the combination of convolution and max pooling layers like we did in our previous example. This is because

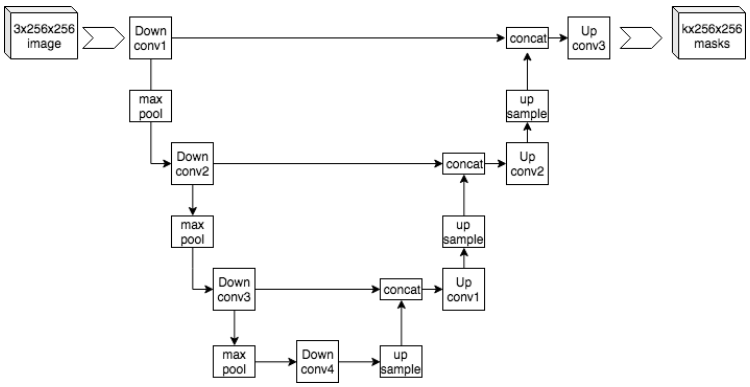


Figure 3.13: Example architecture of U-Net for producing k 256-by-256 image masks for a 256-by-256 RGB image. [40]

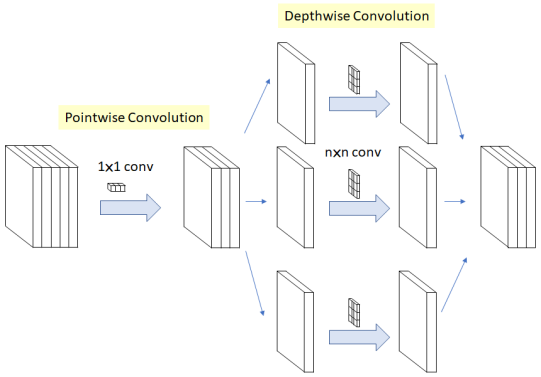


Figure 3.14: The Modified Depthwise Separable Convolution used as an Inception Module in Xception. [37]

this example is also inspired by the Xception architecture, which uses a modified depth-wise separable convolution, which is basically a **point-wise convolution followed by a depth-wise convolution**. An illustration of this type of convolution can be found in Fig. 3.14.

Chapter 4

IVUS Image Segmentation

The problem we will be facing in this project is segmenting IVUS images. Intravascular ultrasound (IVUS) is a catheter-based imaging technology that allows us to see a coronary artery from the inside-out. This unique point-of-view picture, generated in real time, yields more information than other routine imaging methods, such as coronary angiography, or even non-invasive scans.

4.1 How does it work?

IVUS uses echocardiography. Firstly, very high frequency sound waves, called ultrasound, are emitted. Then, these ultrasound waves, which are beyond the range of human hearing, bounce off different types of tissue structures in the body and this echo is then converted into a picture. In the case of IVUS, the transducers are placed on the tip of a catheter. This catheter can be inserted through the coronary arteries. In practice, it becomes a camera that gives us a cross-sectional view, which we can use to detect diseased arteries. Also, the image can be interpreted using a longitudinal view, as we can see in Fig. 4.1.

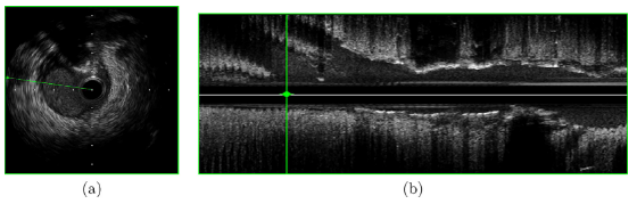


Figure 4.1: Short-axis view of a vessel (a) and longitudinal view of the same pullback (b). The dotted green line in (a) indicates the angular position of the longitudinal view, while the green solid line in (b) identifies the corresponding frame position in the pullback.

These images give us valuable information to recognize the different layers inside the artery.

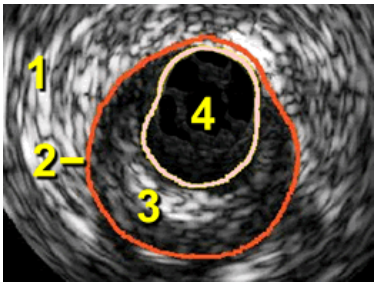


Figure 4.2: Different layers inside the artery.

1. **The adventia**: the outer covering of the artery.
2. **The media**: the actual wall of the artery.
3. **The intima**: a layer of endothelial and other cells that make direct contact with the blood inside the artery – in normal arteries this layer is thin; in diseased arteries (Fig. 4.2) the intima is thickened by plaques or other tissue growth, often eccentric or asymmetrical.
4. **The lumen**: the actual open channel of the artery through which the blood flows.

Therefore, this work aims to explore different ways to automate the process of detecting such layers using Deep Learning algorithms.

4.2 Our goal

The main goal of this work is to explore different deep learning approaches to segment IVUS images. In terms of computer vision, this means that we will perform semantic segmentation of our image to obtain a mask, as we can see in Fig. 3.6.

A mask in image processing is a binary image consisting of zero and non-zero values, so that when we apply this mask on another image of the same size, all the pixels which are zero in the mask are set to zero in the output image. Therefore, masks let us segment an image into two different sections. However, in our case, we want to be able to recognize three different sections: the background, the media/adventitia and the lumen. So we will be using multiclass masks, which are masks with more than 2 values.

4.3 Dataset

We will be using one of the datasets described in the article by Balocco et al. [4]. These datasets are designed to be useful with different approaches that might need to use either a single frame or a multi-frame dataset. There are two datasets publicly available which were obtained using two different ultrasound frequencies: 20 MHz and 40 MHz. The 20 MHz dataset, called the Volcano Dataset, contains two sets (train and test) of IVUS frames of a full pullback at the diastolic

cardiac phase from 10 patients. There are 109 and 326 pullbacks in the training and testing sets respectively. The 40 MHz dataset, called Boston Scientific Dataset, also comes with two predefined sets (train and test) which consist of 19 and 59 IVUS frames respectively. Due to the fact that the number of images available to train is very small, the difficulty to train a good model that can generalize this task is very high. For this project, we will focus on using the first dataset to find a suitable model, this is because there are more images in this dataset and they are separated by patients, also the images in general have more contrast and therefore they are easier to segment.

An IVUS sequence consists of inserting an ultrasound emitter in the arterial vessel and pulling the probe from the distal to the proximal position, this is called *pullback*. In this case, a subset of 5 significative frames has been extracted for each sequence. These 5 frames are extracted around the end-diastolic cardiac phase. Diastole is the instant in the cardiac cycle in which the vessel pulsates less. For this reason, the 5 frames are supposed to be *similar* and this extra information can be exploited to improve the classification of the middle image. We can find an example of one pullback from each dataset in Fig. 4.3.

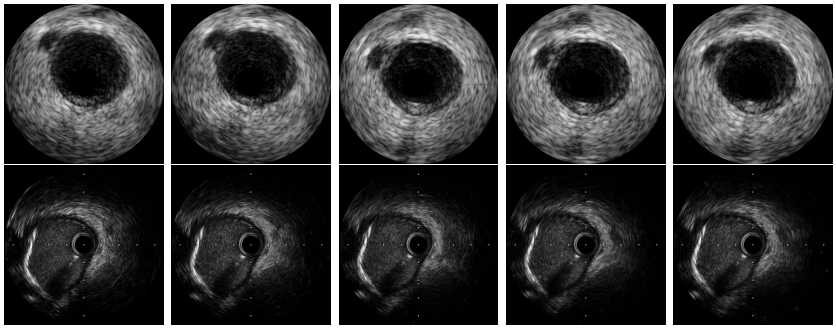


Figure 4.3: Pullback: IVUS sequences of 5 images. Volcano Dataset (20MHz) in the first row. Boston Scientific Dataset (40MHz) second row.

Also, we are given a set of annotations for each IVUS sequence. These annotations have been provided by four clinical experts who are working daily with the specific IVUS echograph brand, and they belong to distinct medical centers. Moreover, two of them repeated the task one week after the first marking. These labels consist of the delineation of both inner wall (lumen contour) and outer wall (media/adventitia contour).

4.3.1 Preprocessing

Firstly, in order to obtain the masks, we draw the annotations as contours and select the pixels that remain inside. After that, we assign a different value to each

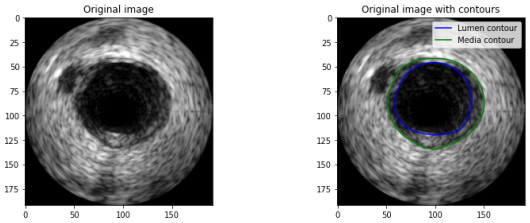


Figure 4.4: IVUS image without and with the lumen and media contours.

pixel depending on which class it belongs to.

We will be working with a custom loss function. This loss function is the Jaccard loss function and the code has been provided by Csurka and Larlus [13] and the **Keras Team** [22]. This function assumes that the labelling and the output of the model are the same, therefore the labels need to be one-hot-encoded.

Since we have three different classes, the encoding of each class is in the following way:

- **Background** pixels: $[1, 0, 0]$
- **Media/adventitia** pixels: $[0, 1, 0]$
- **Lumen** pixels: $[0, 0, 1]$

Also, images have three channels (red, green and blue), so by using this encoding, we can easily visualize the mask. Then, we can interpret this encoding as the probability of a pixel belonging to a class, so that a pixel with value $[0.6, 0.3, 0.1]$ would have the following probabilities: 0.6 of belonging to background, 0.3 of belonging to media/adventitia and 0.1 of belonging to lumen.

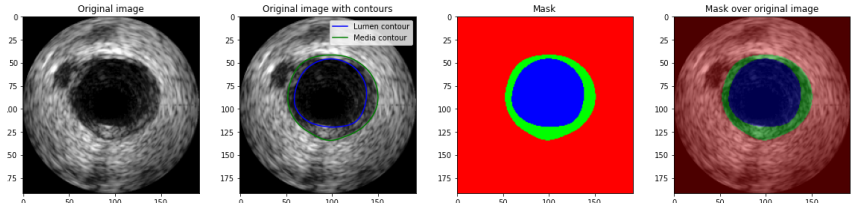


Figure 4.5: Process to obtain a mask and apply it on a sequence.

After separating our training and testing sets, we find that we have images of size 384×384 px for the Volcano Dataset and 512×512 for the Boston Scientific Dataset. With the provided hardware, our batch size cannot be very large since our GPU, which has 6GB of dedicated video memory, will quickly run out of memory. For that reason, we also downsample the images by a factor of 2, so we end up with images of 192×192 px and 256×256.

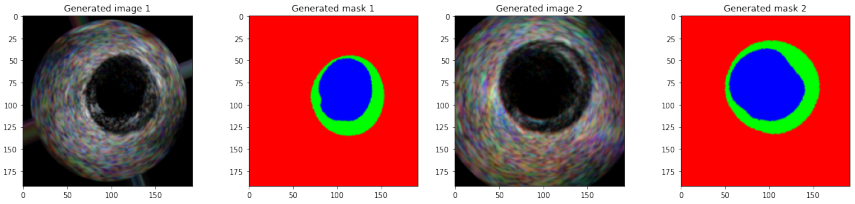


Figure 4.6: Generated images after performing data augmentation (rotations, horizontal and vertical flips and zoom).

4.3.2 Data Augmentation

Clearly, the most notable problem with our dataset is that it has very little data. If we were to train a model with this dataset, the model would quickly overfit. That is because the model would start to **memorize** the training labels for each pixel instead of **learning** how to label them in the image context. In order to avoid this, we need to apply **data augmentation**. The idea is to increase the amount of data by adding slightly modified copies of data from our dataset. For images, the most frequent data augmentation techniques are transformations such as geometric transformations, color modifications, cropping, noise injection, etc. In our case, we will perform the most realistic data augmentations: rotations, horizontal and vertical flips and zoom. So we generate new images like the ones shown in Fig. 4.6.

Another thing we should take into account is the fact that we have 5 images in each pullback. Since the adjacent frames around the middle one might contain valuable information, we will use the middle image and the 2 images adjacent to it as the input of our model. Because our images are in gray scale, we can join these 3 images as if they were one with 3 channels (RGB).

4.4 Model A: First approach

Our first approach to perform semantic segmentation is very simple; the model from the example of a non-sequential deep neural network A.1. In this case, we will use the categorical crossentropy loss function and we will measure the accuracy of every pixel being predicted correctly. The Adam optimizer with a learning rate of 10^{-3} will be used.

The problem we find with this approach is that we have a very high class imbalance. In total, there are 48070656 pixels to classify in our training set: 84.90% of which belong to the background, 6.49% belong to the media/adventitia and 8.60% belong to the lumen. Since we want to focus on labelling correctly the

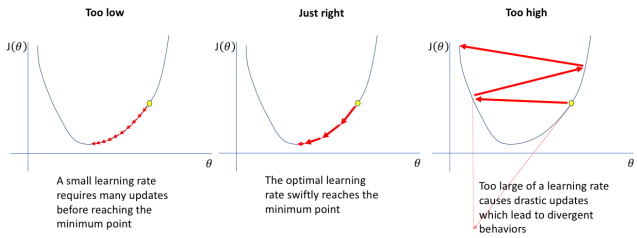


Figure 4.7: Different learning rates behaviour

pixels that belong to the media and the lumen, this first approach might produce bad results.

4.5 Model B: Introducing Jaccard loss

For our second model, we will try to mitigate the effects of the class imbalance by changing the loss function to the Jaccard loss function. As explained in 3.4.1, this statistic is used to determine the similarity of two areas that overlap. Therefore, it gives the same weight to the three different classes. Also, in order to track the progress during training, instead of printing the accuracy, we can use the Dice Similarity Coefficient, also known as Dice Index (DI). Given two sets X and Y the Dice similarity coefficient is defined as

$$DSC(X,Y) = \frac{2|X \cap Y|}{|X| + |Y|}$$

When applied to Boolean data, the DSC can be written as

$$DSC = \frac{2TP}{2TP + FP + FN} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

which coincides with the definition of the f1-score, explained in the results section. So, by using DSC in each class, we can see the real progress during the training phase.

Another thing to consider is that it is not recommended to use a constant learning rate, but to lower it as the training goes on. We can see an illustrative representation of how too big or too low learning rates can affect the loss function in Fig. 4.7. So lowering the learning rate as the function approaches a minimum can have a positive impact.

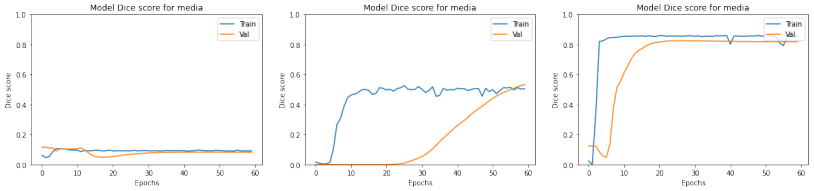


Figure 4.8: Dice score for the media using these architectures, in order: ResNet50, DeepLabV3+, InceptionResNet.

4.6 Model C and D: Introducing Transfer Learning

In this section we introduce two new models. So far, we have been using the three middle images of the pullback as input. Instead, we will now consider using only the middle image as input in order to see how the model behaves. So we will first train with only the middle image for model C and all three images for model D.

In order to improve the classification of each pixel, we can leverage the knowledge of other models that have been trained to classify images, like we explained in 3.5.3 Transfer Learning. Since we are not actually classifying images, but performing semantic segmentation, we need to build U-Net models (Fig. 3.13) out of these pretrained models. This means that we will use a pretrained model as the encoder of our model and connect it to a decoder which will be trained. So, first of all, we need to choose which model works best for our problem.

We have selected three models to choose from: ResNet50, DeepLabV3+ and InceptionResNet. DeepLabV3+ is the only architecture that aims to perform semantic segmentation, the other two models need to have the decoder built separately. In all three cases we freeze the layers of the encoder and train the decoder. After training, the Dice Index in the validation set for background and lumen is very high in all three cases, around 92%. However, we see a very significative difference in the detection of the media. The InceptionResNet outperforms both of them. While the DeepLabV3+ is only able to improve a little bit, the ResNet50 does not improve at all. Fig. 4.8 shows these results. Therefore, we choose InceptionResNet for these models.

4.7 Model E: Adding new labels

As we mentioned before, we have 109 different pullbacks for the training set. However, we have 3 different observations of each pullback. Therefore, we can leverage this fact to triple our training set. This being said, we will have images

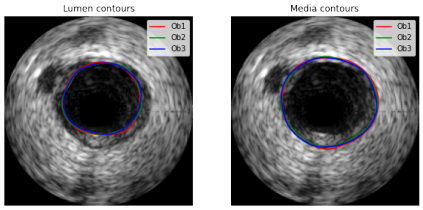


Figure 4.9: Three different observed lumen (right) and media contours (left).

that have a slight different labelling and this may affect positively or negatively to our model. So now, we will have 109 images with different contours, but since these contours were provided by experts, the difference between them will be minimal. A representation of these different contours can be seen in Fig. 4.9.

For this model, we only introduce these new labels in the training set, we do not change any hyperparameter, and we use the InceptionResNet architecture.

4.8 Model F: Elastic Deformations

In recent years, there have been major improvements in CNNs by using elastic deformation in order to augment the training samples. As described in the paper *Best practices for convolutional neural networks applied to visual document analysis* [32], elastic deformations generally produce better results than other affine distortions. Elastic deformations are created by first generating random displacement fields, that is $\forall(x,y)$, $\mathcal{X}(x,y) = \text{rand}(-1,1)$ and $\mathcal{Y}(x,y) = \text{rand}(-1,1)$, where $\text{rand}(-1,1)$ is a random number between -1 and 1, generated with a uniform distribution. The fields $\mathcal{X}$ and $\mathcal{Y}$ are then convolved with a Gaussian of standard deviation σ (in pixels). If σ is large, the resulting values are very small because the random values average 0. If we normalize the displacement field, the field is then close to constant, with a random direction. If σ is small, the field looks like a completely random field after normalization. For intermediate values of σ, the displacement fields look like elastic deformation, where σ is the elasticity coefficient. The displacement fields are then multiplied by a scaling factor α that controls the intensity of the deformation. In Fig. A.25 we can see the outputs of different σ values. In our case, we will use $\sigma \approx 20$px as it produces good elastic and realistic deformations.

Now, we use this elastic deformation as our data augmentation method to generate as many images as we want. So the last model we propose uses the same architecture as before, InceptionResNet, but adding this change on the training augmentation.

Chapter 5

Results

In this section we present the results obtained by the different models we proposed in the last chapter using both datasets. In order to evaluate the models we use some common metrics as well as some metrics presented by Balocco et al. [4]. We compute some classification metrics with the help of a library called `sklearn`. These are:

- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$, where TP, TN, FP and FN are True Positives, True Negatives, False Positives and False Negatives respectively.
- Precision: $\frac{TP}{TP+FP}$, where TP and FP are the number of True Positives and False Positives respectively.
- Recall or sensitivity (True Positive Rate): $\frac{TP}{TP+FN}$, where TP and FN are the number of True Positives and False Negatives respectively.
- Specificity (True Negative Rate): $\frac{TN}{TN+FP}$
- F1 score: the harmonic mean of precision and recall, therefore it gives more weight to the lower values. $\frac{2}{recall^{-1}+precision^{-1}} = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$, where FN is the number of False Negatives.

Moreover, we evaluate the generated masks with some measures useful to compare the similarity of different sample sets:

- Jaccard index: consider two sets A and B, the jaccard index is defined as $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$, where $||$ is the cardinality.
- Hausdorff distance: we use it to measure how far away each set of points are. It is defined as $d_H(X,Y) = max\{\sup_{x \in X} d(x,Y), \sup_{y \in Y} d(y,X)\}$ and, in this case, its unit is a **pixel**. However, by knowing the distance each pixel represents in each dataset, we transform it into **millimetres**.

Furthermore, each model has been trained using K-Fold Cross Validation with $K = 5$ predetermined folds, so the models have the same training, validation and testing set in each fold. And the models with new labels have the same images in every set but with the added labels, so no training image is in the training and validation set at the same time. Lastly, the data augmentation has only been applied to the training set.

5.1 Model A

Using this naive approach, we obtain a pretty good first result. Indeed, we get an overall accuracy of 98% for our training and validation set, we can see the graphs of the loss and accuracy during the training phase in Fig. A.1. Also, note that there might be an initialization problem and that's why it overfits at the beginning of the training.

These results do look well but they are overshadowed by the imbalance between classes that we explained before. To have a more in-depth report of the actual accuracy of each class, we can evaluate the model on our testing set and print the confusion matrix. As we can see in Fig. A.2, even though the overall accuracy is nearly 95%, only 57% of the media surface is predicted correctly and nearly 30% of our media prediction is wrong. In the case of lumen, 96.55% of our lumen prediction is correct but we only predict correctly 78.83% of the lumen surface. This confusion matrix was tested using the model trained with the last fold.

The metrics of the K-Fold Cross Validation on the Volcano Dataset are as follows:

Table 5.1: Metrics of the model A evaluated on the Volcano Dataset test set. Each metric has the average and standard deviation over each fold.

	background	media	lumen
Accuracy	0.97 (0.0017)	0.96 (0.0013)	0.98 (0.0004)
Precision	0.97 (0.0015)	0.71 (0.0103)	0.97 (0.0036)
Recall	1.00 (0.0007)	0.62 (0.0146)	0.81 (0.0083)
F1-score	0.98 (0.0010)	0.66 (0.0120)	0.88 (0.0037)
Specificity	0.81 (0.0010)	0.98 (0.0007)	1.00 (0.0003)
Jaccard Measure	0.97 (0.0019)	0.50 (0.0134)	0.79 (0.0060)
Hausdorff Distance	0.71 (0.4368)	0.74 (0.4490)	0.48 (0.3948)

The **F1-score** in lumen and media classes are very poor, so we should try to incentivize our model to classify those better. From the **Jaccard Measure** we clearly see that the media and lumen are highly misrepresented. For instance, only 71% of the media pixels from all images coincide with the real media. Also, the **Hausdorff Distance** tells us that there are a lot of outliers in our mask predictions. Even though the model is able to identify most of the classes correctly, we see that there are points which lie really far away from the ground truth set. This distance averages nearly a tenth of the image width or height.

Also, we compute the metrics of the *K*-Fold Cross Validation on the Boston Scientific Dataset:

Table 5.2: Metrics of the model A evaluated on the Boston Scientific test set. Each metric has the average and standard deviation over each fold.

	background	media	lumen
Accuracy	0.93 (0.0025)	0.90 (0.0030)	0.96 (0.0022)
Precision	0.94 (0.0041)	0.58 (0.0166)	0.89 (0.0110)
Recall	0.97 (0.0020)	0.54 (0.0246)	0.77 (0.0302)
F1-score	0.96 (0.0014)	0.56 (0.0137)	0.83 (0.0134)
Specificity	0.78 (0.0185)	0.95 (0.0044)	0.99 (0.0020)
Jaccard Measure	0.92 (0.0026)	0.39 (0.0131)	0.71 (0.0193)
Hausdorff Distance	1.83 (0.7126)	2.29 (1.0040)	1.66 (0.9730)

As expected, the results are much worse than in the other dataset, due to the low amount of data and the low contrast.

Now, the first thing we should focus on is trying to solve the Hausdorff Distance issue, namely the problem with the outliers. So we proceed to evaluate the next model.

5.2 Model B

In this model we have changed the loss function to Jaccard. In addition, we also use a weights initializer for our model to see if the model does not overfit during so many epochs at the beginning of the training. We use the Glorot normal initializer, also called Xavier normal initializer. This initializer draws, for each weight w, a number from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = \sqrt{\frac{2}{inputs + outputs}}$.

The results of the training phase can be found in Fig. A.3. Just like before, we can see that there is still a problem. During the first 100 epochs the model is

overfitting, it probably only classifies the pixels as background. However, when the model cannot minimize the loss function anymore, it starts to learn to classify the other classes and starts getting better results.

As we did in our first approach, we evaluate the model in the testing set and plot the confusion matrix, Fig. A.4. In this case, we used the model trained in the last fold since it had the best results for the validation set. We can compare it with the confusion matrix of model A and see an improvement of the sensitivity of the media, going from 63.83% in model A, to 72.76% in model B.

Also, we can print the same metrics as before and to compare this model with model A, we add how much the average has increased or decreased.

Table 5.3: Metrics of the model B evaluated on the Volcano Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to the last model.

	background	media	lumen
Accuracy	0.96 (0.0161) -0.01	0.95 (0.0156) -0.01	0.98 (0.0064) 0.00
Precision	0.96 (0.0186) -0.01	0.30 (0.3621) -0.41	0.84 (0.0458) -0.13
Recall	0.99 (0.0025) -0.01	0.30 (0.3648) -0.32	0.87 (0.0261) +0.06
F1-score	0.98 (0.0084) 0.00	0.30 (0.3634) -0.36	0.87 (0.0261) -0.01
Specificity	0.73 (0.1373) -0.08	0.99 (0.0090) +0.01	0.98 (0.0110) -0.02
Jaccard Measure	0.95 (0.0161) -0.02	0.24 (0.2889) -0.26	0.78 (0.0417) -0.01
Hausdorff Distance	0.77 (0.4130) +0.06	0.61 (0.3975) -0.13	0.76 (0.7956) +0.28

Table 5.4: Metrics of the model B evaluated on the Boston Scientific Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to the last model.

	background	media	lumen
Accuracy	0.92 (0.0155) -0.01	0.90 (0.0104) 0.00	0.95 (0.0082) -0.01
Precision	0.92 (0.0215) -0.02	0.24 (0.2915) -0.34	0.81 (0.1010) -0.08
Recall	0.99 (0.0063) +0.02	0.22 (0.2693) -0.32	0.85 (0.0839) +0.08
F1-score	0.95 (0.0086) +0.01	0.23 (0.2799) -0.33	0.82 (0.0124) -0.01
Specificity	0.64 (0.1014) -0.14	0.98 (0.0232) +0.03	0.97 (0.0212) -0.02
Jaccard Measure	0.90 (0.0157) -0.02	0.16 (0.1960) -0.23	0.69 (0.0180) -0.02
Hausdorff Distance	2.06 (0.8063) +0.23	2.21 (0.9624) -0.08	1.73 (1.0451) +0.07

We can find an interesting result in the first table, Table 5.3. The media classification is very poor and has a high standard deviation. This is caused by the fact that the model only learned to classify some pixels as media in 2 folds out of the 5 folds we performed. In Fig. A.3, we only showed the metrics of the last fold, the other 3 folds in which the model did not learn to classify the media pixels, there is a dice score of 0 in all epochs for both the training and validation sets. We could try to explore why this is happening, but since this is the most simple model, we will move onto trying to improve a more promising one.

5.3 Model C

In this model we introduced transfer learning and changed the architecture. We are now using InceptionResNet. We have also changed the input, instead of using 3 frames of the pullback, we are only using one. It is going to be interesting to compare these results with model D, which uses 3 frames and, therefore, more information. We can see in Fig. A.5 that the model converged better during the training phase, even though there is some overfitting at the beginning. This might be caused by the fact that the dataset is very small and this the model *memorizes* all the labels in the beginning of the training, but eventually starts learning how to do the segmentation.

Now, let's show the table of metrics of model C and compare them with the first model, which gave the best results so far.

Table 5.5: Metrics of the model C evaluated on the Volcano Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to model A, because model B has very bad results.

	background	media	lumen
accuracy	0.98 (0.0014) +0.01	0.96 (0.0027) 0.00	0.98 (0.0020) 0.00
Precision	0.98 (0.0044) +0.01	0.73 (0.0430) +0.02	0.95 (0.0171) -0.02
Recall	0.99 (0.0032) -0.01	0.75 (0.0618) +0.13	0.87 (0.0409) +0.06
F1-score	0.99 (0.0007) +0.01	0.73 (0.0208) +0.07	0.90 (0.0157) +0.02
Specificity	0.90 (0.0289) +0.09	0.98 (0.0055) +0.01	1.00 (0.0017) 0.00
Jaccard Measure	0.97 (0.0014) 0.00	0.58 (0.0262) +0.08	0.82 (0.0258) +0.03
Hausdorff Distance	0.59 (0.3985) -0.12	1.17 (1.0204) +0.43	0.54 (0.5346) +0.05

Table 5.6: Metrics of the model C evaluated on the Boston Scientific Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to model A, because model B has very bad results.

	background	media	lumen
Accuracy	0.93 (0.0056) 0.00	0.92 (0.0046) +0.02	0.97 (0.0025) +0.01
Precision	0.93 (0.0098) -0.01	0.68 (0.0281) +0.10	0.91 (0.0093) +0.02
Recall	0.99 (0.0045) +0.02	0.49 (0.0675) -0.05	0.83 (0.0212) +0.06
F1-score	0.96 (0.0032) 0.00	0.57 (0.0429) +0.01	0.87 (0.0113) +0.04
Specificity	0.74 (0.0417) -0.04	0.97 (0.0057) +0.02	0.99 (0.0015) 0.00
Jaccard Measure	0.92 (0.0058) 0.00	0.40 (0.0423) +0.01	0.76 (0.0174) +0.05
Hausdorff Distance	1.97 (0.8528) +0.14	2.26 (1.0670) -0.03	1.51 (1.1448) -0.15

This model clearly has a better performance than model A or B, even though it is using less information as input. Despite the improvements, there are still some results that do not look good. For example, the Hausdorff Distance has a lot of variance which means that the results are not very reliable.

5.4 Model D

This model is identical to model C, but it uses the 2 adjacent frames of the middle image from the pullback. Since it receives more information as input, we expect to get, at least, equal results.

Table 5.7: Metrics of the model D evaluated on the Volcano Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to model C.

	background	media	lumen
Accuracy	0.98 (0.0009) 0.00	0.97 (0.0011) +0.01	0.99 (0.0006) +0.01
Precision	0.98 (0.0015) 0.00	0.79 (0.0255) +0.06	0.93 (0.0162) -0.02
Recall	0.99 (0.0010) 0.00	0.72 (0.0290) -0.03	0.91 (0.0042) +0.04
F1-score	0.99 (0.0005) 0.00	0.75 (0.0092) +0.02	0.92 (0.0042) +0.02
Specificity	0.90 (0.0097) 0.00	0.99 (0.0025) +0.01	0.99 (0.0017) -0.01
Jaccard Measure	0.98 (0.0009) +0.01	0.60 (0.0119) +0.02	0.85 (0.0073) +0.03
Hausdorff Distance	0.55 (0.3691) -0.04	0.67 (0.4632) -0.50	0.51 (0.5350) -0.04

Table 5.8: Metrics of the model D evaluated on the Boston Scientific Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to model C.

	background	media	lumen
Accuracy	0.94 (0.0028) +0.01	0.92 (0.0028) 0.00	0.97 (0.0022) 0.00
Precision	0.94 (0.0076) +0.01	0.70 (0.0302) +0.02	0.90 (0.0153) -0.01
Recall	0.98 (0.0062) -0.01	0.54 (0.0550) +0.05	0.85 (0.0305) +0.02
F1-score	0.96 (0.0015) 0.00	0.61 (0.0284) +0.04	0.88 (0.0115) +0.01
Specificity	0.79 (0.0318) +0.05	0.97 (0.0068) 0.00	0.99 (0.0027) 0.00
Jaccard Measure	0.93 (0.0028) +0.01	0.44 (0.0290) +0.04	0.79 (0.0179) +0.03
Hausdorff Distance	1.91 (0.8532) -0.06	3.02 (1.9234) +0.76	1.77 (1.4571) +0.26

The results shown in Tables 5.7 and 5.8 look much better than the model before. Indeed, the adjacent frames we provided are very useful to the neural network to determine the lumen and media areas. The most notable improvement is in the Hausdorff Distance of the media, which has decreased by nearly 50%. Overall, the classification of the media is the hardest task to perform. We still see very low results on Jaccard Measure for this class.

5.5 Model E

In this model we added two new labels for every image in the dataset, from two different observations. The architecture is the same as model D, as well as, the data augmentation used. The metrics of this model are as follows:

Table 5.9: Metrics of the model E evaluated on the Volcano Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to model D.

	background	media	lumen
Accuracy	0.98 (0.0006) 0.00	0.97 (0.0018) 0.00	0.99 (0.0015) 0.00
Precision	0.98 (0.0014) 0.00	0.75 (0.0186) -0.04	0.97 (0.0086) +0.04
Recall	0.99 (0.0012) 0.00	0.76 (0.0145) +0.04	0.85 (0.0257) -0.06
F1-score	0.99 (0.0003) 0.00	0.75 (0.0117) 0.00	0.90 (0.0115) -0.02
Specificity	0.89 (0.0100) -0.01	0.98 (0.0019) -0.01	1.00 (0.0008) +0.01
Jaccard Measure	0.98 (0.0006) 0.00	0.61 (0.0151) +0.01	0.82 (0.0190) -0.03
Hausdorff Distance	0.54 (0.3620) -0.01	0.73 (0.4894) +0.06	0.55 (0.5173) +0.04

Table 5.10: Metrics of the model E evaluated on the Boston Scientific Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to model D.

	background	media	lumen
Accuracy	0.94 (0.0018) 0.00	0.93 (0.0022) +0.01	0.97 (0.0024) 0.00
Precision	0.95 (0.0063) +0.01	0.73 (0.0377) +0.03	0.89 (0.0366) -0.01
Recall	0.98 (0.0072) 0.00	0.56 (0.0610) +0.02	0.89 (0.0300) +0.04
F1-score	0.96 (0.0011) 0.00	0.63 (0.0288) +0.02	0.89 (0.0068) +0.01
Specificity	0.80 (0.0254) +0.01	0.97 (0.0074) 0.00	0.98 (0.0065) -0.01
Jaccard Measure	0.93 (0.0021) 0.00	0.46 (0.0304) +0.02	0.80 (0.0110) +0.01
Hausdorff Distance	1.89 (0.8272) -0.02	2.53 (1.2432) -0.49	1.84 (1.5375) +0.07

In general, the metrics are slightly worse than our previous model for the Volcano Dataset, but there is not much difference. Maybe the data augmentation used is not letting us improve our model.

In the case of the Boston Scientific Dataset, this model performed much better. This means that it could be further improved by feeding new, but similar, data. So we might need to improve our data augmentation method. Let's make another experiment on this model.

5.5.1 Comparing data augmentations

In all the tests we have performed so far, we only used the most, apparently, physically realistic data augmentation types for this data: zoom, horizontal & vertical flips and rotations. They have been useful to avoid overfitting when training the model, but are they all necessary and physically realistic? In order to answer this question, we can train the same architecture with the same hyperparameters using different data augmentations and evaluate the model produced with the Volcano Dataset test set.

The metrics we are most interested in are the **recall** and **f1-score**. So we plot a graph comparing all these data augmentation types. We can find these graphs in Fig. 5.1 and Fig. 5.2. The results are interesting, if we look at the media's recall, the most meaningful data augmentations are rotations and horizontal & vertical flips, whereas for the f1-score these data augmentation yield worse results than width & height shifts, and zoom is the most meaningful one. The fact that rotations and flips have better results in the recall and worse results in f1-score implies that the precision metric is low for these data augmentation, because the f1-score is the

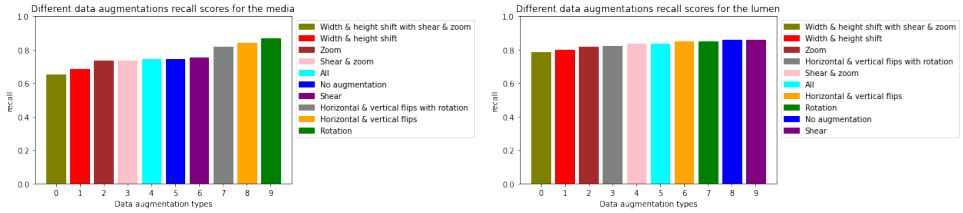


Figure 5.1: Recall metric after training the InceptionResNet model using one image and different data augmentations.

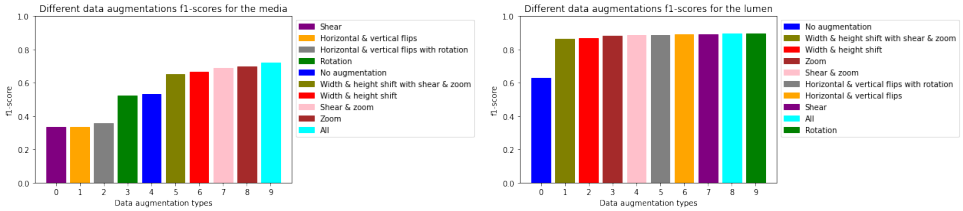


Figure 5.2: F1-score metric after training the InceptionResNet model using one image and different data augmentations.

harmonic mean between the precision and recall. This means that rotating and flipping makes the model have a low amount of false negatives, but the amount of false positives is greater compared with the model using width & height shifts. This might be caused by the fact that width & height shifts cause the catheter not be in the center of the image, which will never be the case, and that leads to more false negatives.

For the lumen, we see that all data augmentations produce mostly the same results.

5.6 Model F

In this last approach we introduced a new type of data augmentation: elastic deformation. Actually, with this method we can generate as many different images as we want, since the algorithm is not deterministic. The parameters chosen for this method were based on trying to represent the images as physically realistic as possible. This means that they might not be optimal for our model, since we didn't choose them based on the model performance. The architecture of the model is the same as the other models: InceptionResNet with the encoder pretrained with the ImageNet dataset. The results of this model are shown in the next tables.

Table 5.11: Metrics of the model F evaluated on the Volcano Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to model D.

	background	media	lumen
Accuracy	0.98 (0.0013) 0.00	0.96 (0.0034) -0.01	0.98 (0.0031) -0.01
Precision	0.98 (0.0028) 0.00	0.73 (0.0518) -0.06	0.94 (0.0200) +0.01
Recall	0.99 (0.0015) 0.00	0.70 (0.0375) -0.02	0.83 (0.0573) -0.08
F1-score	0.99 (0.0007) 0.00	0.71 (0.0175) -0.04	0.88 (0.0246) -0.04
Specificity	0.86 (0.0189) -0.04	0.98 (0.0005) -0.01	1.00 (0.0021) +0.01
Jaccard Measure	0.97 (0.0014) -0.01	0.55 (0.0209) -0.05	0.79 (0.0392) -0.06
Hausdorff Distance	0.66 (0.4122) +0.11	0.98 (0.6051) +0.31	0.75 (0.7272) +0.24

Table 5.12: Metrics of the model F evaluated on the Boston Scientific Dataset test set. Each metric has the average and standard deviation over each fold. The rightmost number indicates how much the metrics' average have increased or decreased compared to model E.

	background	media	lumen
Accuracy	0.92 (0.0034) -0.02	0.91 (0.0039) -0.02	0.97 (0.0020) 0.00
Precision	0.92 (0.0052) -0.03	0.67 (0.0336) -0.06	0.89 (0.0242) 0.00
Recall	0.98 (0.0025) 0.00	0.40 (0.0679) -0.16	0.86 (0.0249) -0.03
F1-score	0.95 (0.0019) -0.01	0.50 (0.0512) -0.13	0.87 (0.0075) -0.02
Specificity	0.71 (0.0234) -0.09	0.98 (0.0063) +0.01	0.98 (0.0044) 0.00
Jaccard Measure	0.91 (0.0035) -0.02	0.33 (0.0457) -0.13	0.77 (0.0118) -0.03
Hausdorff Distance	2.15 (0.8976) +0.26	2.49 (1.0395) -0.04	1.74 (1.2628) -0.10

Clearly these results are quite poor compared to model D. Note that the Hausdorff Distances are very high, which means that there are many points in our lumen and media masks that are quite far from the actual lumen and media sections. However, in the case of the Volcano Dataset, if we have a look at the metrics during the training phase in Fig. A.11, we see that the Dice Index for media is over 0.80 and for lumen is much higher, whereas in the table we see that for the test set we have a Dice Index average of 0.71 with a low standard deviation. This fact leads us to another question: are the images of the 3 patients in the training set much more different from the other 7 patients?

5.6.1 Training with different patients

As we have explained, the Volcano Dataset contains IVUS sequences from 10 different patients. Our training set has the sequences from 3 patients, while the other 7 are used as the test set. For that reason, we do one last experiment which is doing a 5-Fold Cross Validation over pairs of patients:

- Fold 1. Training set: patients 1 and 2. Validation set: patient 3. Test set: patients 4 to 10.
- Fold 2. Training set: patients 3 and 4. Validation set: patient 5. Test set: patients 1, 2 and 5 to 10.
- Fold 3. Training set: patients 5 and 6. Validation set: patient 7. Test set: patients 1 to 4 and 7 to 10.
- Fold 4. Training set: patients 7 and 8. Validation set: patient 9. Test set: patients 1 to 6, 7 and 8.
- Fold 5. Training set: patients 9 and 10. Validation set: patient 1. Test set: patients 1 to 8.

The results are shown in Table 5.13, we can see the standard deviation on the precision and recall is quite high. This means that depending on which patients we train our model with, the model performance changes quite drastically. In general, the previous results in Table 5.11 are better than the average on this table. So we can assume that our training dataset is in fact a good representation of the sequences of different patients, even though there might be lots of differences between them.

Furthermore, in section A.3.2 we have shown several figures comparing the models produced from each fold and model F. Since every image belongs to a training set of some model, we have opted to show 5 different figures, where in each figure we show an image belonging to different patients and, therefore, to a training set of different models. Note that the best prediction is always from the model that used the image in its training, obviously. Nevertheless, we can see that the first model always gives a pretty good segmentation.

Table 5.13: Metrics of the model F after performing 5-Fold Cross Validation using different patients from the Volcano Dataset for training, validation and test set as explained above. Each metric has the average and standard deviation over each fold.

	background	media	lumen
Accuracy	0.96 (0.0057)	0.95 (0.0052)	0.98 (0.0058)
Precision	0.98 (0.0098)	0.63 (0.0699)	0.93 (0.0416)
Recall	0.98 (0.0152)	0.75 (0.0967)	0.82 (0.0446)
F1-score	0.98 (0.0034)	0.68 (0.0165)	0.87 (0.0350)
Specificity	0.89 (0.0576)	0.97 (0.0121)	0.99 (0.0035)
Jaccard Measure	0.96 (0.0065)	0.51 (0.0187)	0.78 (0.0542)
Hausdorff Distance	0.64 (0.4332)	0.79 (0.0411)	0.48 (0.2424)

5.7 Summary

In this chapter we have presented the results and the thought process of each metric used for all of models as we explored them. Here, we want to summarize the results obtained. Clearly the best model is **model D**. In the Appendix section A.3 we have added some predictions for both the Volcano Dataset and Boston Scientific Dataset. They can be easily compared. Looking at the first figure (Fig. A.13) we see that all models do segment the lumen correctly but the only one segmenting the media with good results is model D. Then we have a Fig. A.14, where all the models show a pretty accurate segmentation. Finally, Fig. A.15 is an edge case, we see a little shadow near the centre that confuses the models and makes them think there is a lumen section there and we end up with this two non-connected lumen masks. So, for the most common IVUS sequences, model D works pretty fine, but there are still some cases where it clearly underperforms.

Table 5.14: Metrics of each model evaluated on the Volcano Dataset. These metrics are Jaccard Measure (JM), Hausdorff Distance (HD) [mm] and Dice Index (DI) separated by lumen and media. The metrics in bold represent the best scores.

	lumen			media		
	JM	HD	DI	JM	HD	DI
Model A	0.79 (0.0060)	0.48 (0.3948)	0.88 (0.0037)	0.50 (0.0134)	0.74 (0.4490)	0.66 (0.0120)
Model B	0.78 (0.0417)	0.76 (0.7956)	0.87 (0.0261)	0.24 (0.2889)	**0.61 (0.3975)**	0.30 (0.3634)
Model C	0.82 (0.0258)	0.54 (0.5346)	0.90 (0.0157)	0.58 (0.0262)	1.17 (1.0204)	0.73 (0.0208)
Model D	**0.85 (0.0073)**	**0.51 (0.5350)**	**0.92 (0.0042)**	**0.60 (0.0119)**	0.67 (0.4632)	**0.75 (0.0092)**
Model E	0.82 (0.0190)	0.55 (0.5173)	0.90 (0.0115)	0.61 (0.0151)	0.73 (0.4894)	0.75 (0.0117)
Model F	0.79 (0.0392)	0.75 (0.7272)	0.88 (0.0246)	0.55 (0.0209)	0.98 (0.6051)	0.71 (0.0175)

Table 5.15: Metrics of each model evaluated on the Boston Scientific Dataset. These metrics are Jaccard Measure (JM), Hausdorff Distance (HD) [mm] and Dice Index (DI) separated by lumen and media. The metrics in bold represent the best scores.

	lumen			media		
	JM	HD	DI	JM	HD	DI
Model A	0.71 (0.0193)	1.66 (0.9730)	0.83 (0.0134)	0.39 (0.0131)	2.29 (1.0040)	0.56 (0.0137)
Model B	0.69 (0.0180)	1.73 (1.0451)	0.82 (0.0124)	0.16 (0.1960)	**2.21 (0.9624)**	0.23 (0.2799)
Model C	0.76 (0.0174)	**1.51 (1.1448)**	0.87 (0.0113)	0.40 (0.0423)	2.26 (1.0670)	0.57 (0.0429)
Model D	0.79 (0.0179)	1.77 (1.4571)	0.88 (0.0115)	0.44 (0.0290)	3.02 (1.9234)	0.61 (0.0284)
Model E	**0.80 (0.0110)**	1.84 (1.5375)	**0.89 (0.0068)**	**0.46 (0.0304)**	2.53 (1.2432)	**0.63 (0.0288)**
Model F	0.77 (0.0118)	1.74 (1.2628)	0.87 (0.0075)	0.33 (0.0457)	2.49 (1.0395)	0.50 (0.0512)

Table 5.16: Characteristics of all the methods we have tested our datasets on.

	Model architecture	Nº frames per pullback	Training set size	Loss function	Data augmentation
Model A	Keras U-Net	3 frames	109 pullbacks	Categorical Crossentropy	Rotation, zoom and vertical & horizontal flips
Model B	Keras U-Net (initialized with Glorot)	3 frames	109 pullbacks	Categorical Crossentropy	Rotation, zoom and vertical & horizontal flips
Model C	InceptionResNet	1 frame	109 pullbacks	Jaccard	Rotation, zoom and vertical & horizontal flips
Model D	InceptionResNet	3 frames	109 pullbacks	Jaccard	Rotation, zoom and vertical & horizontal flips
Model E	InceptionResNet	3 frames	327 pullbacks	Jaccard	Rotation, zoom and vertical & horizontal flips
Model F	InceptionResNet	3 frames	327 pullbacks	Jaccard	Elastic deformations

Chapter 6

Conclusions

This final section is divided into two parts. The first part explains the conclusions given by the experience and results from this project and the second part proposes new unexplored approaches that could help either solve the problem or, at least, improve its current solution.

6.1 Project Conclusions

The aim of this project was to learn, study, develop and validate a Deep Learning model capable of automatically segment IVUS images, and all these goals have been met. We have presented the process from learning how and why neural networks work, to actually building some models to tackle a real problem in semantic segmentation of medical imaging. In the last section we had a general view of the results achieved by the models we developed. We have focused on improving the model and the training data by using some advanced techniques. We have improved quite well, from a naive model to a more complex one with great results. However, there is still room for improvement.

The most notable improvement was to leverage transfer learning from other models. The architectures we tested on were ResNet50, DeepLabV3+ and InceptionResNet. InceptionResNet was the only one learning to classify the media, which is the hardest section to segment. This forced us to choose this architecture in order to keep exploring and we focused on improving the training data afterwards.

A visual comparison of some results can be seen in the section A.3. The first two figures show some usual IVUS images where we can see subtle differences between the results and model D is the more precise one. Nevertheless, we also showed an edge case in Fig. A.15, where we can see an example of the InceptionResNet struggling to segment correctly. Even though the lumen is usually well

segmented, we can see that there are two non-connected sets of points, which is something that should be avoided by trying to minimize the Hausdorff Distance. Then we have the media, which is clearly the most difficult to segment because the images sometimes have these shadows that confuse our model and, as we saw in our last experiment, depending on which patients we train, the results can vary a lot.

In conclusion, we have achieved to segment the lumen section of an IVUS sequence with good results and the media section with promising results, but still with lots of improvements to perform. The architecture we employed was InceptionResNet pretrained with the ImageNet Dataset and connected like a U-Net to a decoder. We can show how the results from our best model look compared to some of the state-of-the-art approaches:

Table 6.1: Comparison of the performance of different IVUS images segmentation algorithms evaluated on the dataset described at [4]. The evaluation measures are Jaccard Measure (JM), Hausdorff Distance (HD) [mm] and Dice Index (DI). The average and standard deviation or an interval are provided. The bold metrics indicate the results of our proposed model.

	JM_{lumen}	HD_{lumen}	DI_{lumen}	JM_{media}	HD_{media}	DI_{media}
Nandamuri et al. (2019) [28]	0.95 ± 0.03	0.17 ± 0.07	-	0.97 ± 0.01	0.16 ± 0.09	-
Yang et al. (2018) [39]	0.90 (0.06)	0.26 (0.25)	-	0.86 (0.11)	0.48 (0.44)	-
Farahi et al. (2018) [17]	0.91 (0.04)	0.22 (0.12)	-	0.83 (0.15)	0.50 (0.45)	-
Yang et al. (2019) [38]	0.87 (0.07)	0.82 (0.61)	-	0.86 (0.09)	1.07 (0.72)	-
Balakrishna et al. (2018) [3]	0.7982 (-)	-	0.8846 (-)	0.8085 (-)	-	0.8825 (-)
Lo Vercio et al. (2019) [25]	0.83 (0.10)	0.57 (0.31)	-	0.88 (0.08)	0.32 (0.25)	-
`Model D` (our approach)	**0.85 (0.0073)**	**0.51 (0.5350)**	**0.92 (0.0042)**	**0.60 (0.0119)**	**0.67 (0.4632)**	**0.75 (0.0092)**
Bargsten et al. (2021) [6]	-	0.629 ± 0.123	93.52 ± 1.05	-	0.825 ± 0.116	79.11 ± 1.15
Downe et al. (2008) [15]	0.77 (0.09)	0.47 (0.22)	-	0.74 (0.17)	0.76 (0.48)	-

With different approaches we are sure that new DL models can be found, so we encourage the research on this real problem, which could help thousands of people.

6.2 Further work

After exploring some methods of DL semantic segmentation, we suggest new approaches that could help improve future work on this problem:

- Focus on class imbalance. Nearly 86% of the pixels to classify belong to the background class, this makes our model learn to prioritize the classification

of this class. However, we actually want to focus on having a good segmentation of the other classes. That's why we suggest to use weights in the loss function to incentivize the classification of lumen and media pixels.

- Use other loss functions that also minimize the Hausdorff distance, because the media and the lumen should always be connected sets [21].
- Test different data augmentation methods. For this problem, artificial shadows in the images could be used to simulate the shadows from bifurcations or side vessels. Also, elastic deformation could be analyzed more thoroughly.
- Explore other architectures using other data augmentation, or deeper architectures such as ResNet-101, which require more training data since it has more weights to optimize.
- Instead of freezing the layers in the encoder, we can use the pretrained architecture as a weights initializer and train the model as one, or with the encoder separately.
- Our images were circular because we used the polar representation. It is possible to train the same model but using Cartesian coordinates, as shown in Fig. 4.1, which can lead to better results.
- Perform a statistical test to check if the differences between the images of the patients are statistically significant.

Appendix A

Tests and results

In this appendix we provide code and images of metrics from training the models and some tests performed.

A.1 Code for building a Keras model

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

Example of a sequential deep neural network [12]

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1	(MaxPooling2 (None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 10)	16010

Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0

```
inputs = keras.Input(shape=img_size + (3,))

### [First half of the network: downsampling inputs] ###

# Entry block
x = layers.Conv2D(32, 3, strides=2, padding="same")(inputs)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)

previous_block_activation = x  # Set aside residual

# Blocks 1, 2, 3 are identical apart from the feature depth.
for filters in [64, 128, 256]:
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(filters, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(filters, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    # Project residual
    residual = layers.Conv2D(filters, 1, strides=2, padding="same")(
        previous_block_activation
    )
    x = layers.add([x, residual])  # Add back residual
```

```
    previous_block_activation = x  # Set aside next residual

### [Second half of the network: upsampling inputs] ###

for filters in [256, 128, 64, 32]:
    x = layers.Activation("relu")(x)
    x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.Activation("relu")(x)
    x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)

    x = layers.UpSampling2D(2)(x)

    # Project residual
    residual = layers.UpSampling2D(2)(previous_block_activation)
    residual = layers.Conv2D(filters, 1, padding="same")(residual)
    x = layers.add([x, residual])  # Add back residual
    previous_block_activation = x  # Set aside next residual

# Add a per-pixel classification layer
outputs = layers.Conv2D(num_classes, 3, activation="softmax", padding=
    "same")(x)

# Define the model
model = keras.Model(inputs, outputs)
```

Example of a non-sequential deep neural network [12]

A.2 Training metrics

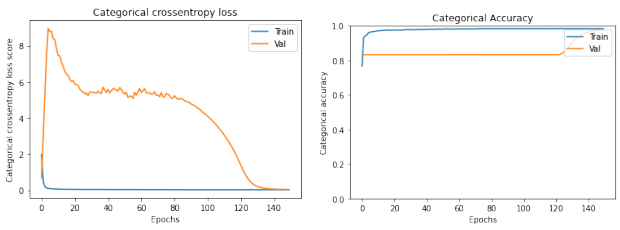


Figure A.1: Model loss and accuracy during training phase of model A.

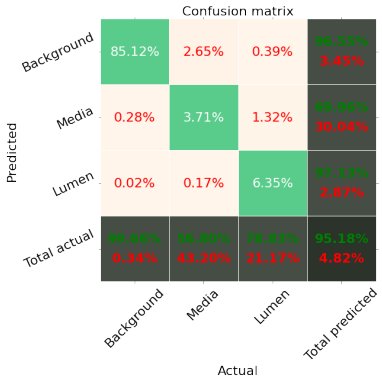


Figure A.2: Confusion matrix of model A

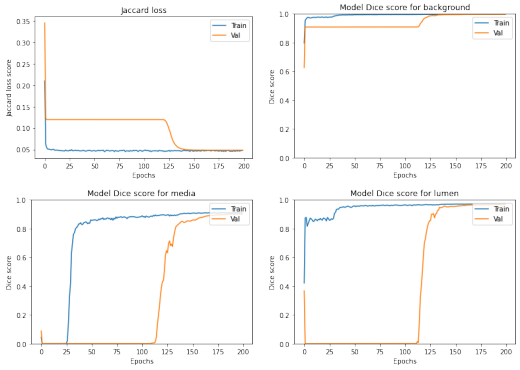


Figure A.3: Loss function and the metrics evaluated during the training phase of model B.

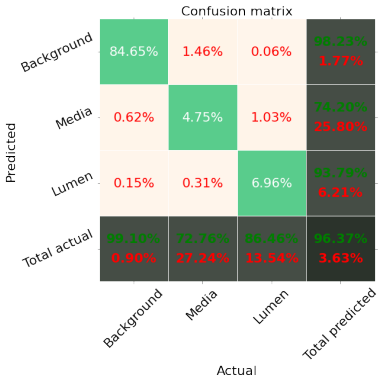


Figure A.4: Confusion matrix of model B

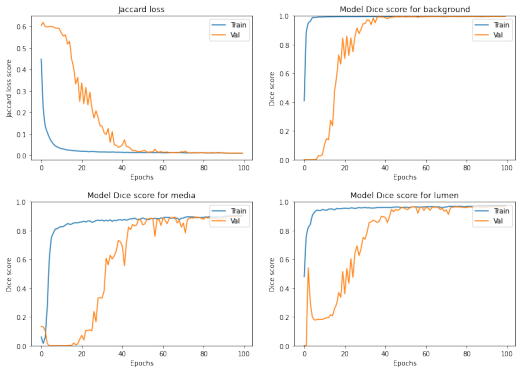


Figure A.5: Loss function and the metrics evaluated during the training phase of model C.

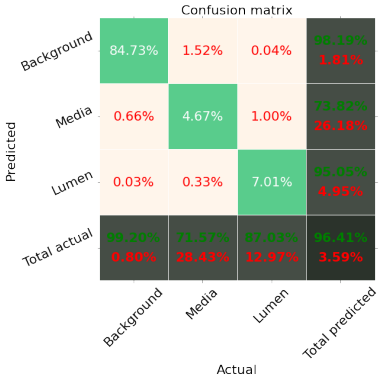


Figure A.6: Confusion matrix of model C

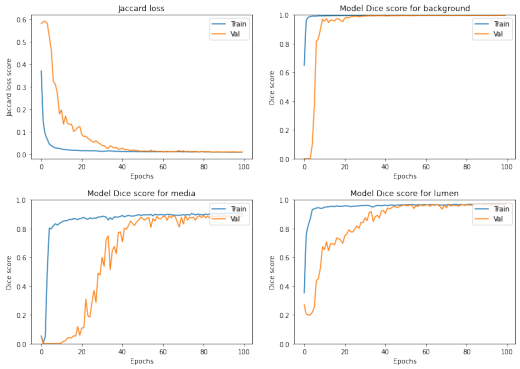


Figure A.7: Loss function and the metrics evaluated during the training phase of model D.

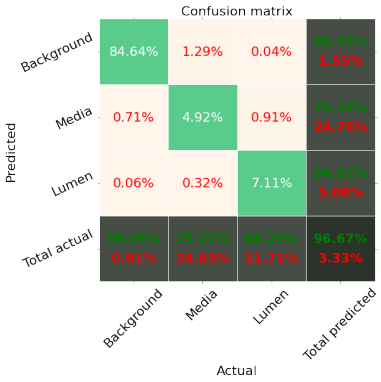


Figure A.8: Confusion matrix of model D

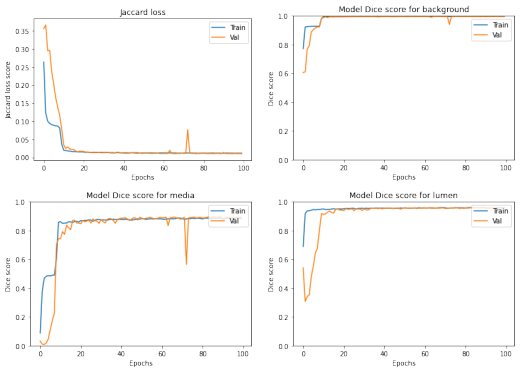


Figure A.9: Loss function and the metrics evaluated during the training phase of model E.

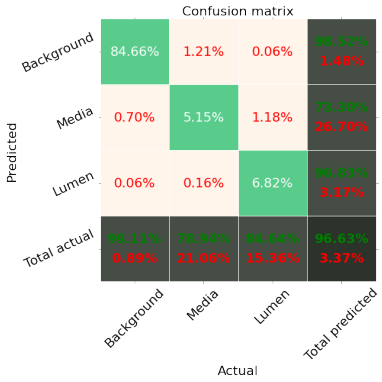


Figure A.10: Confusion matrix of model E

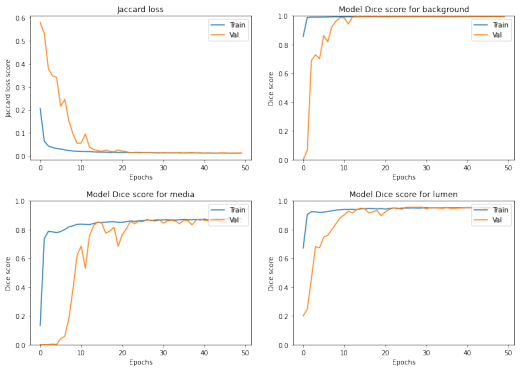


Figure A.11: Loss function and the metrics evaluated during the training phase of model F.

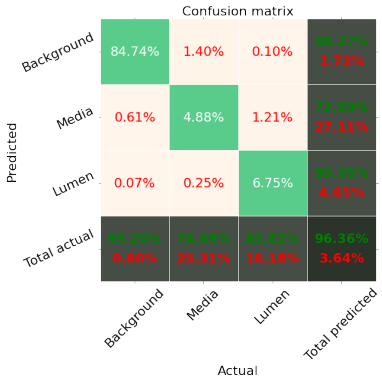


Figure A.12: Confusion matrix of model F

A.3 Predictions

In this section we provide some predictions to compare the performance of the proposed models. This section is divided into three subsections: Volcano Dataset, Volcano Dataset by patients and Boston Scientific Dataset. As their names suggest, there are IVUS sequences from the different datasets, except for the section Volcano Dataset by patients where we can visualize the predictions from model F trained with different patients as explained in the results. Since we used our entire dataset, there is no available test image to see which model performs better. So we opted for showing 5 different sets of images, the first one from patient 2, the next from patient 4, and so on. So the model with best results will always be the one who used the image in the training set. However, we can see how the other models perform in those cases. The training set is indicated in each set of images. Furthermore, we have also reproduced those predictions for Model F.

We have provided both easy and edge cases of segmentation. We consider edge cases the ones that give worse results for the best models. Note that the lumen and media are supposed to be connected sets, but there are some cases where the model confuses a shadow and mispredicts where the lumen or media are.

As we have seen in the results section, the Boston Scientific Dataset is much more difficult to segment. This difficulty comes from the fact that the training size is very low (19 IVUS sequences) and the images are darker and therefore have lower contrast.

A.3.1 Volcano Dataset

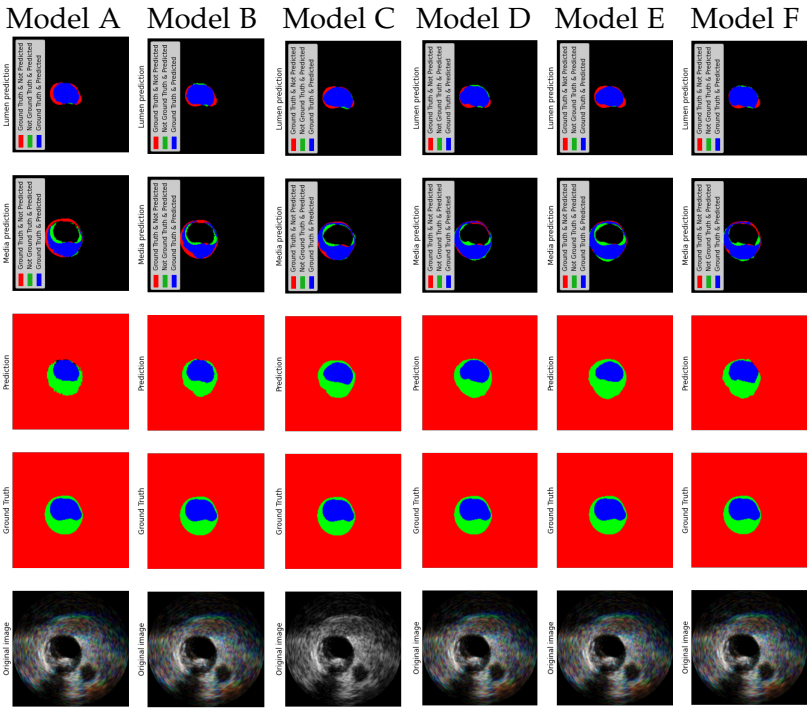


Figure A.13: Predictions of each model, from A to F, of the same frame. The last two images represent the overlap between the masks.

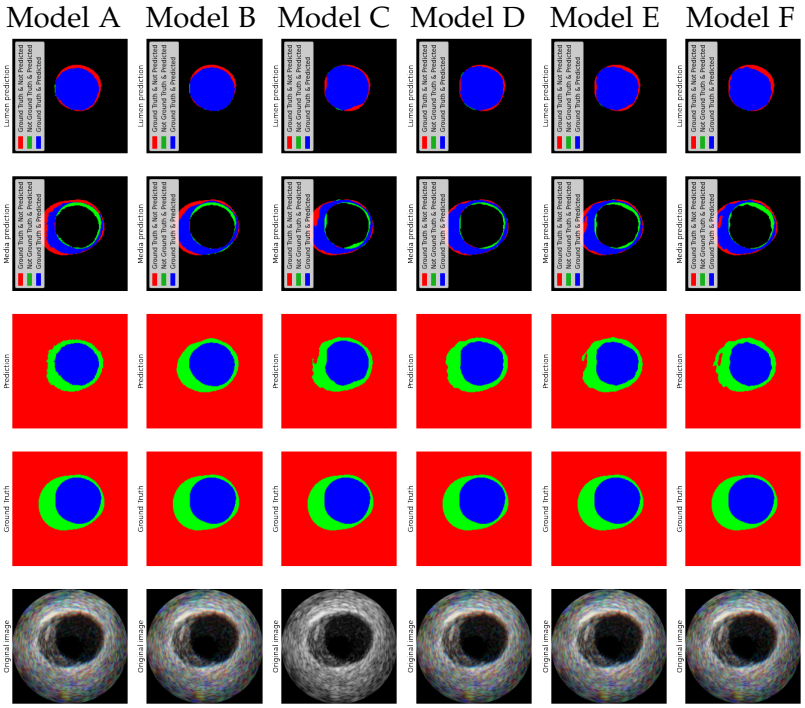


Figure A.14: Predictions of each model, from A to F, of the same frame. The last two images represent the overlap between the masks.

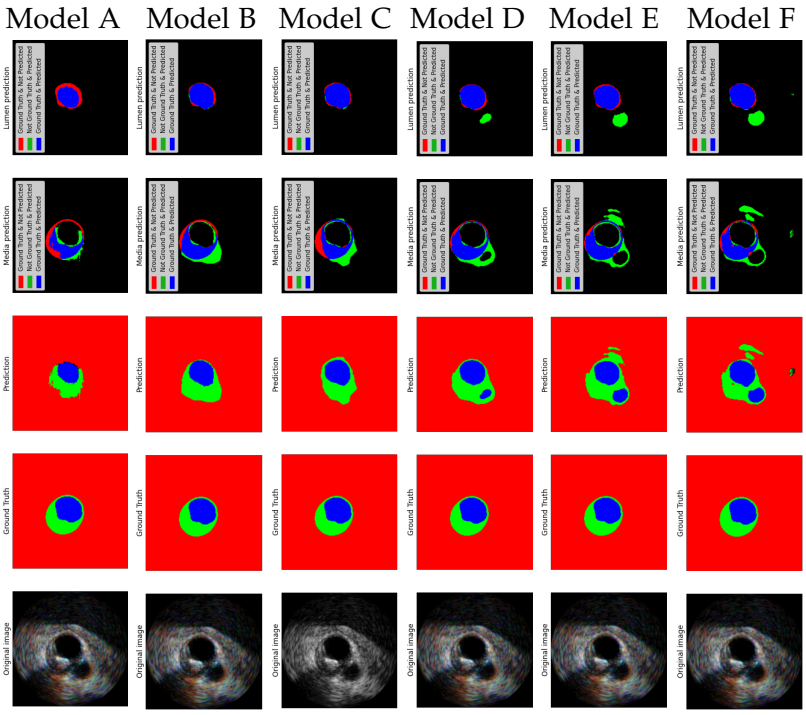


Figure A.15: Predictions of each model, from A to F, of the same frame. The last two images represent the overlap between the masks. This is an edge case because there is a shadow very close to the center.

A.3.2 Volcano Dataset by patients

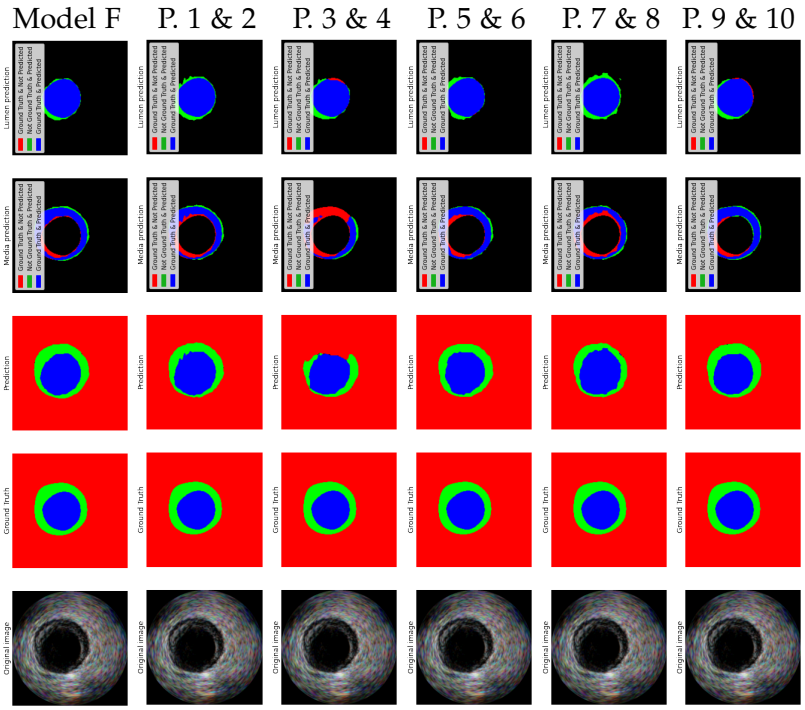


Figure A.16: Predictions of the same frame from model F trained with images from different pages. The last two images represent the overlap between the masks. This is an edge case because there is a shadow very close to the center. The frame is from patient 2.

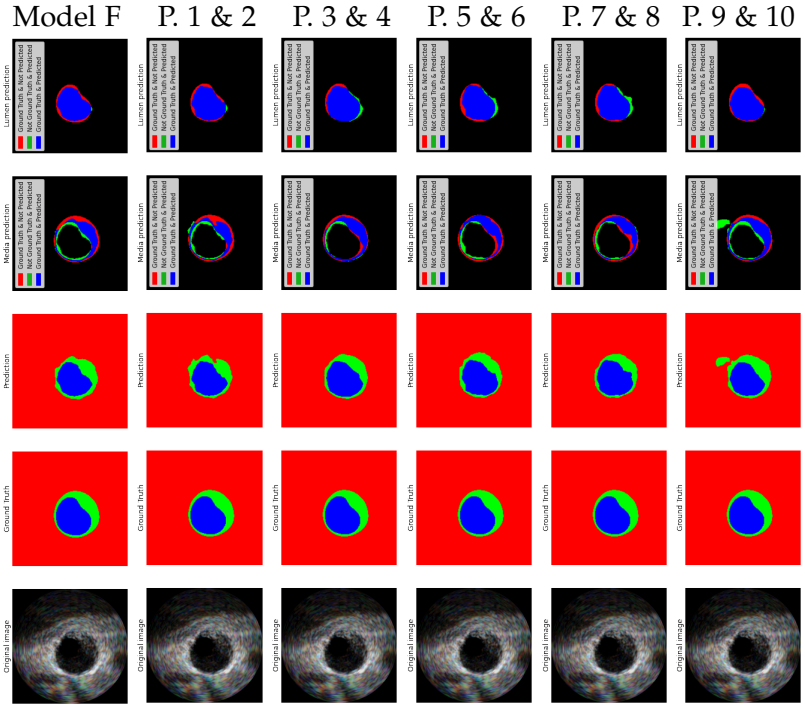


Figure A.17: Predictions of the same frame from model F trained with images from different pages. The last two images represent the overlap between the masks. This is an edge case because there is a shadow very close to the center. The frame is from patient 4.

Figure A.18: Predictions of the same frame from model F trained with images from different pages. The last two images represent the overlap between the masks. This is an edge case because there is a shadow very close to the center. The frame is from patient 6.

Figure A.19: Predictions of the same frame from model F trained with images from different pages. The last two images represent the overlap between the masks. This is an edge case because there is a shadow very close to the center. The frame is from patient 8.

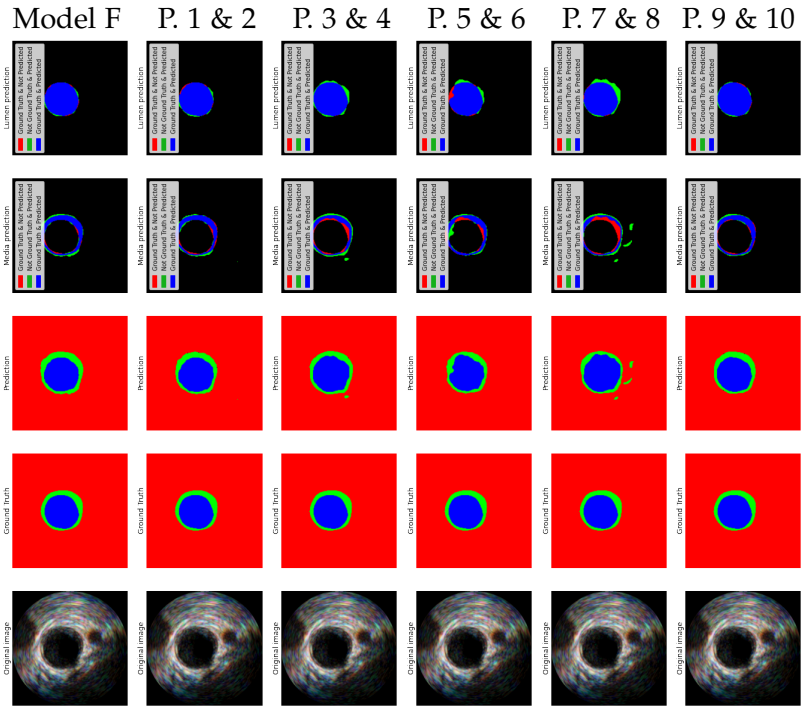


Figure A.20: Predictions of the same frame from model F trained with images from different pages. The last two images represent the overlap between the masks. This is an edge case because there is a shadow very close to the center. The frame is from patient 10.

A.3.3 Boston Scientific Dataset

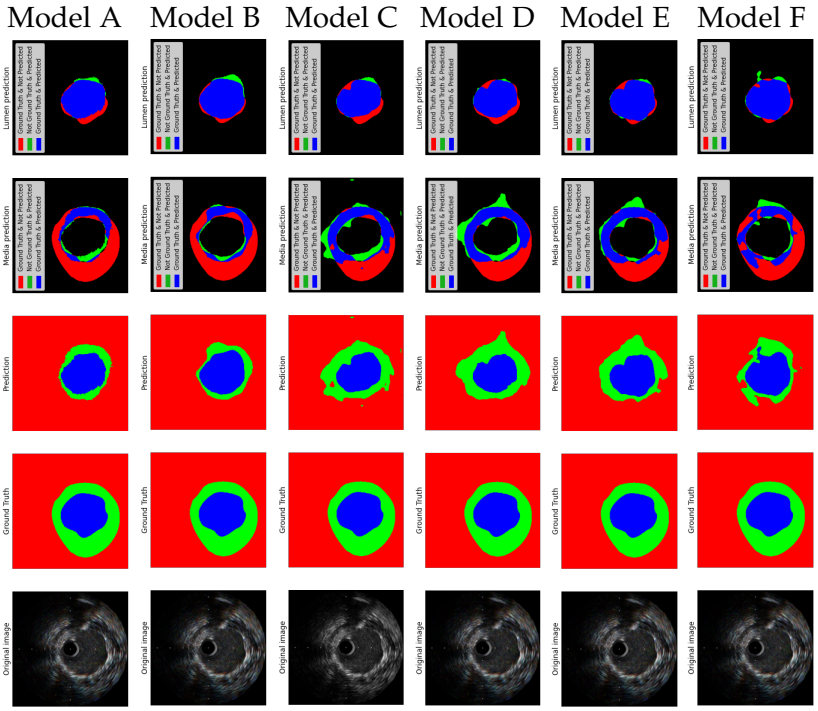


Figure A.21: Predictions of each model, from A to F, of the same frame. The last two images represent the overlap between the masks.

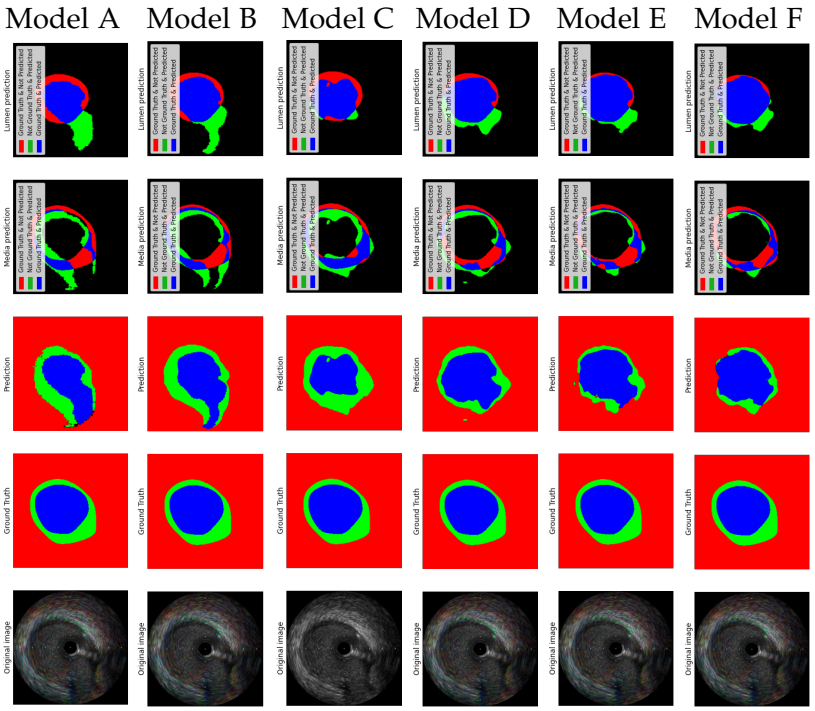


Figure A.22: Predictions of each model, from A to F, of the same frame. The last two images represent the overlap between the masks.

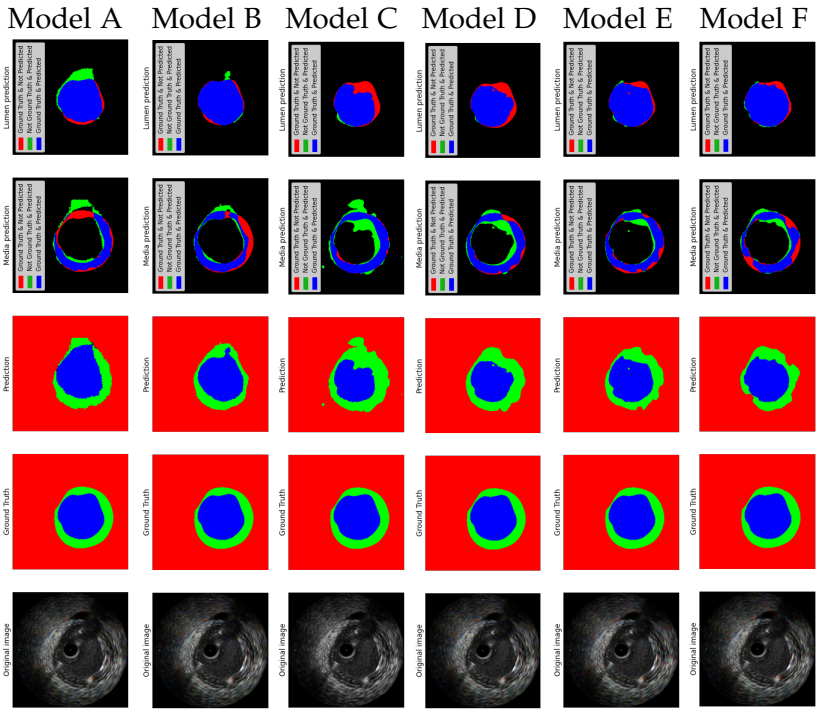


Figure A.23: Predictions of each model, from A to F, of the same frame. The last two images represent the overlap between the masks.

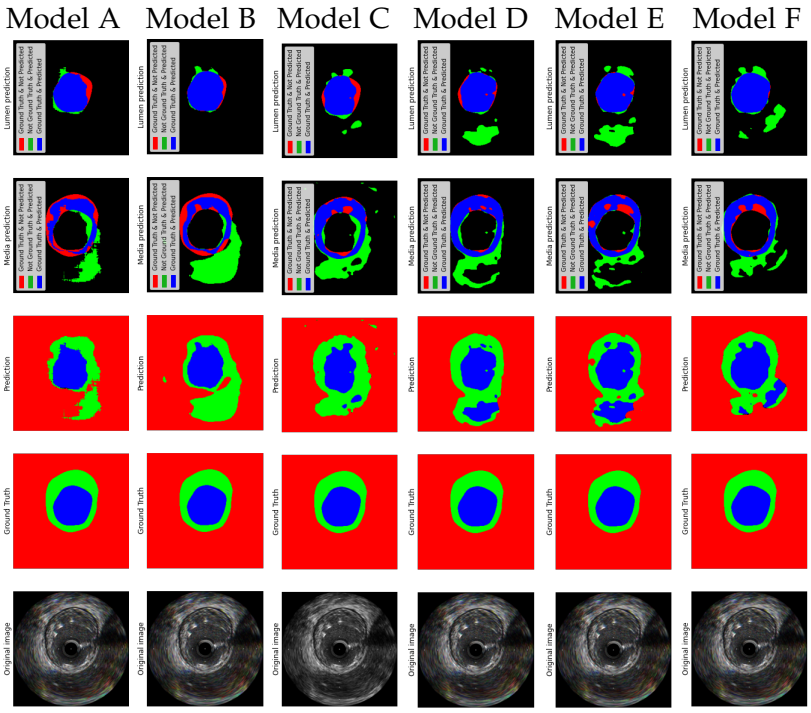


Figure A.24: Predictions of each model, from A to F, of the same frame. The last two images represent the overlap between the masks. This is one of the hardest images to segment.

A.4 Elastic Deformations

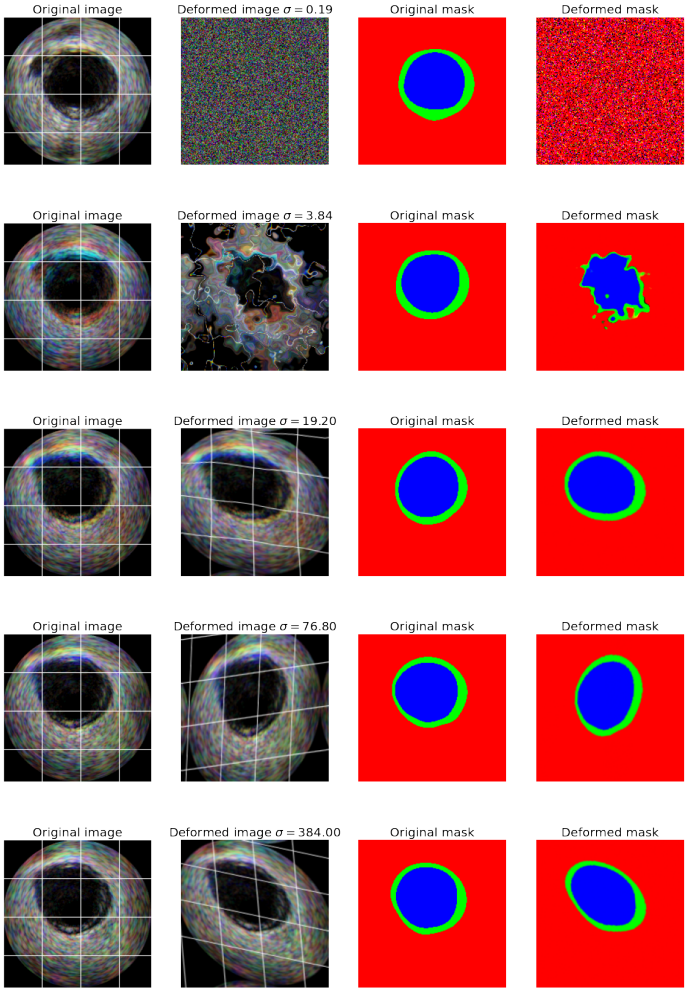


Figure A.25: Elastic deformation algorithm as explained in [32] with different σ values for the Gaussian filter. The algorithm is explained at section 4.8.

Bibliography

[1] Angioplasty.Org, "Intravascular ultrasound (ivus)," February 2013. [Online]. Available: http://ptca.org/ivus/ivus.html

[2] R. Bajaj, X. Huang, Y. Kilic, A. Ramasamy, A. Jain, M. Ozkor, V. Tufaro, H. Safi, E. Erdogan, P. W. Serruys, J. Moon, F. Pugliese, A. Mathur, R. Torii, A. Baumbach, J. Dijkstra, Q. Zhang, and C. V. Bourantas, "Advanced deep learning methodology for accurate, real-time segmentation of high-resolution intravascular ultrasound images," *International Journal of Cardiology*, vol. 339, pp. 185–191, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167527321010287

[3] C. Balakrishna, S. Dadashzadeh, and S. Soltaninejad, "Automatic detection of lumen and media in the ivus images using u-net with vgg16 encoder," 2018.

[4] S. Balocco, "Standardized evaluation methodology and reference database for evaluating ivus image segmentation," *Comput Med Imaging Graph*, vol. 38, no. 2, pp. 70–90, 2014.

[5] S. Bansal, "Cnn architectures : Vgg, resnet, inception + tl," November 2018. [Online]. Available: https://www.kaggle.com/shivamb/cnn-architectures-vgg-resnet-inception-tl

[6] L. Bargsten, S. Raschka, and A. Schlaefer, "Capsule networks for segmentation of small intravascular ultrasound image datasets," *International Journal of Computer Assisted Radiology and Surgery*, pp. 1–12, 2021.

[7] V. Basu, "Transfer learning and unet to segment rocks on moon," March 2019. [Online]. Available: https://www.kaggle.com/basu369victor/transferlearning-and-unet-to-segment-rocks-on-moon

[8] A. Bonner, "Convolutional neural networks and image classification," February 2019. [Online]. Available: https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb

[9] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," 2016.

[10] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017.

[11] F. Chollet, "Transfer learning & fine-tuning," March 2020. [Online]. Available: https://keras.io/guides/transfer_learning/

[12] ——, "Code examples," July 2021. [Online]. Available: https://keras.io/examples

[13] G. Csurka and D. Larlus, "What is a good evaluation measure for semantic segmentation?" vol. 26, 01 2013.

[14] T.-D. Do, M.-T. Duong, Q.-V. Dang, and M.-H. Le, "Real-time self-driving car navigation using deep neural network," in *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, 2018, pp. 7–12.

[15] R. Downe, A. Wahle, T. Kovarnik, H. Skalicka, J. Lopez, J. Horak, and M. Sonka, "Segmentation of intravascular ultrasound images using graph search and a novel cost function," in *Proc. 2nd MICCAI workshop on computer vision for intravascular and intracardiac imaging*. Citeseer, 2008, pp. 71–79.

[16] F. Falah, O. Rahmati, M. Rostami, E. Ahmadisharaf, I. N. Daliakopoulos, and H. R. Pourghasemi, "14 - artificial neural networks for flood susceptibility mapping in data-scarce urban areas," in *Spatial Modeling in GIS and R for Earth and Environmental Sciences*, H. R. Pourghasemi and C. Gokceoglu, Eds. Elsevier, 2019, pp. 323–336. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128152263000144

[17] M. Faraji, I. Cheng, I. Naudin, and A. Basu, "Segmentation of arterial walls in intravascular ultrasound cross-sectional images using extremal region selection," *Ultrasonics*, vol. 84, pp. 356–365, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0041624X17306625

[18] C. Hansen, "Optimizers explained - adam, momentum and stochastic gradient descent," October 2019. [Online]. Available: https://mlfromscratch.com/optimizers-explained/

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[20] IBM, "Neural networks," August 2020. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks

[21] D. Karimi and S. E. Salcudean, "Reducing the hausdorff distance in medical image segmentation with convolutional neural networks," *IEEE Transactions on Medical Imaging*, vol. 39, no. 2, pp. 499–513, 2020.

[22] Keras, "Keras community contributions: Jaccard distance," https://github.com/keras-team/keras-contrib/blob/master/keras_contrib/losses/jaccard.py, 2018.

[23] ——, "About keras," July 2021. [Online]. Available: https://keras.io/about

[24] H. W. Kurt Hornik, Maxwell Stinchcombe, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989. [Online]. Available: https://cognitivemedium.com/magic_paper/assets/Hornik.pdf

[25] L. Lo Vercio, M. del Fresno, and I. Larrabide, "Lumen-intima and media-adventitia segmentation in ivus images using supervised classifications of arterial layers and morphological structures," *Computer Methods and Programs in Biomedicine*, vol. 177, pp. 113–121, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169260718318224

[26] D. McNeela, "The universal approximation theorem for neural networks," March 2017. [Online]. Available: https://mcneela.github.io/machine_learning/2017/03/21/Universal-Approximation-Theorem.html

[27] E.-I. Medium, "Brain and artificial neural networks: Differences and similarities," June 2020. [Online]. Available: https://medium.com/@eraiitk/brain-and-artificial-neural-networks-differences-and-similarities-1d337fe50168

[28] S. Nandamuri, D. China, P. Mitra, and D. Sheet, "Sumnet: Fully convolutional model for fast segmentation of anatomical structures in ultrasound volumes," in *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*, 2019, pp. 1729–1732.

[29] S. Patrikar, "Batch, mini batch & stochastic gradient descent," October 2019. [Online]. Available: https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

[30] M. Ranzato, Y. Taigman, L. Wolf, and M. Yang, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. [Online].

Available: https://research.facebook.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/

[31] O. O. D. Science, "Using the cnn architecture in image processing," January 2020. [Online]. Available: https://medium.com/@ODSC/using-the-cnn-architecture-in-image-processing-65b9eb032bdc

[32] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pp. 958–963, 2003.

[33] P. Sinha, Y. Wu, I. Psaromiligkos, and Z. Zilic, "Lumen amp; media segmentation of ivus images via ellipse fitting using a wavelet-decomposed subband cnn," in *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2020, pp. 1–6.

[34] J. Stewart, G. Manmathan, and P. Wilkinson, "Primary prevention of cardiovascular disease: A review of contemporary guidance and literature," *JRSM Cardiovasc Dis*, vol. 6, p. 2048004016687211, Jan. 2017.

[35] TensorFlow, "Image segmentation," October 2021. [Online]. Available: https://www.tensorflow.org/tutorials/images/segmentation

[36] ——, "Tensorflow," July 2021. [Online]. Available: https://www.tensorflow.org

[37] S.-H. Tsang, "Review: Xception — with depth-wise separable convolution, better than inception-v3 (image classification)," September 2018. [Online]. Available: https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568

[38] J. Yang, M. Faraji, and A. Basu, "Robust segmentation of arterial walls in intravascular ultrasound images using dual path u-net," *Ultrasonics*, vol. 96, pp. 24–33, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0041624X18308059

[39] J. Yang, L. Tong, M. Faraji, and A. Basu, "Ivus-net: An intravascular ultrasound segmentation network," *arXiv preprint arXiv:1806.03583*, 2018.

[40] M. Yazdani, "Example architecture of u-net," October 2021. [Online]. Available: https://en.wikipedia.org/wiki/U-Net