UCLouvain

École polytechnique de Louvain

# Contrastive learning for annotation enrichment and automatic classification of pulmonary embolisms

Author: **Florine Thiry**
Supervisors: **Christophe De Vleeschouwer, Benoît Macq**
Readers: **Eliott Brion, Renaud Detry**
Academic year 2020–2021
Master [120] in Electrical Engineering

# Abstract

Pulmonary embolism (PE) is a disease that can lead to death if not treated quickly. Although early diagnosis is critical, it is also challenging and sometimes missed. Therefore, deep learning methods are starting to be used for automatic PE detection. However, most common methods are limited by their need for a large and accurately labeled dataset.
This master thesis develops two main axes to overcome this limitation. The first one aims to complete a partially annotated dataset, while the second one studies a novel method – contrastive learning - that achieves promising results with fewer labeled data. Both axes are developed in an annotation enrichment task, and this last one is implemented with both supervised and contrastive learning. This task involves training a network with fewer and fewer data to estimate the minimum number of samples needed to annotate the others with accuracy.
The results, although still theoretical, are encouraging. With supervised learning, the annotation work is almost reduced by half. Only 52.97% of the dataset needs to be labeled or verified while the network completes the annotation of the other half with perfect accuracy (reached considering the labels of this remaining part of the dataset). Contrastive learning allows improving this result by achieving a percentage of required labeled data of 49.34%.
This thesis shows that contrastive learning can outperform supervised learning and leads the way to further experiments more focused on PE detection purposes.

# Acknowledgments

This master thesis has been a consequent work for which I benefited from the support of multiple people. Through this intervention, I would like to express my gratitude to the latter.

First, I want to thank the team that accompanied me throughout this journey. Prof. Christophe De Vleeschouwer, Prof. Benoît Macq, Eliott Brion, and Victor Joos de ter Beerst, thank you all for the support you showed me until the end, the thoughtful advice you gave me, and your availability all along with the development of this work.

Second, my gratitude goes to my parents, Vincent Thiry and Anathalie Mukundwa, and my sisters, Sophie and Elodie Thiry, for their unconditional support in every step of this thesis, and especially against every encountered obstacle. Thank you for always making me believe in my abilities.

Third, I would like to thank Michel Henry and Maud Cawet for their spontaneous and quality proofreading. A work of this scope is always enriched with external opinions.

A big thank you also to Alexandre Englebert, who has agreed to meet me without second thoughts to give me some key pieces of information I could not have obtained by myself.

Then, I would like to express my gratitude to Renaud Detry. Thank you for the interest you have shown in my work by accepting to be a jury member.

Finally, last but not least, my housemates and friends Fanny Rouxhet, Axelle Wilmet, Lola Tamigneaux, and Oriane de Leuze. Thank you for listening to all my complaints and for your presence alongside me throughout this work.

# Contents

# List of Figures

# List of Tables

CHAPTER 1

# Introduction

Pulmonary embolism (PE) is «the third most common cause of cardiovascular death worldwide after stroke and heart attack» [1]. It is due to blood clots traveling through the veins that clog the pulmonary artery tree, as illustrated in Figure 1.1.

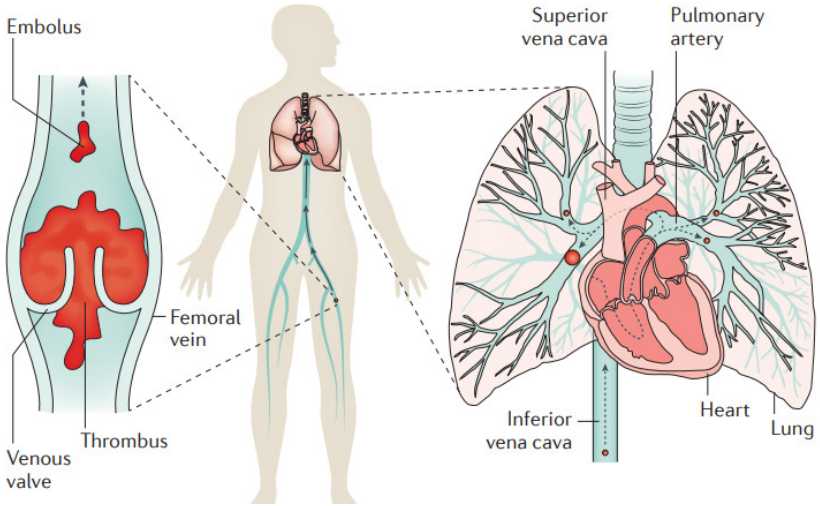
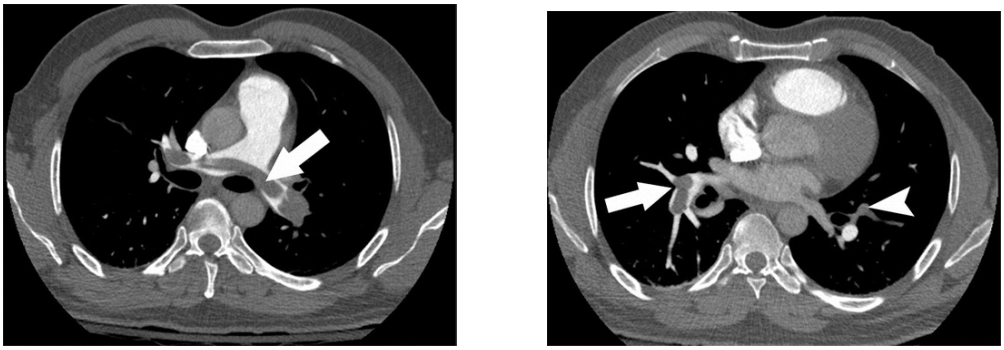



Figure 1.1: Mechanisms of PE [2].

Early detection is, among other things, critical in the management of PE [2]. However, the diagnosis of PE is a challenging task because the symptoms are nonspecific [1, 3] and inconsistently present [4]. As a result, PE diagnosis is part of «the most frequently missed or delayed» diagnoses [5].
When patients are susceptible to suffer from PE, chest imaging studies are performed. The definitive diagnosis can be made from these since, among other things [6], PE can be directly identified, as shown in Figure 1.2.



(a) Central PE. (b) Right-sided PE and Left-sided PE.

Figure 1.2: Examples of pulmonary embolisms [6].

However, the detailed analysis of hundreds of images is time-consuming for radiologists [6]. Therefore, applications based on deep learning could have significant use in this context. This thesis is thus part of the research about automatic PE detection.

Recently, the artificial intelligence (AI) domain has taken a considerable leap forward with the rise of deep learning. This new subset of AI has brought a new area filled with opportunities for improvement in domains such as computer vision, speech recognition, robotics, AI gaming, etc. [7].

Deep learning is a field that has grown tremendously over the last ten years. Even though the researches started a few decades ago [8], researchers did not yet have the means to turn the background ideas into experiments. The reason is that the training of deep networks requires significant computing power, which the technology could not provide at the time. Hence, it was when computers became faster that deep learning experiments began. Then, they took off again with the development of GPUs (Graphics Processing Units) [9, 10].
In 2012, a supervised deep network called AlexNet obtained the first significant results [11]. Since then, deep learning has attracted more and more people and has improved a lot over the years [12].

The underlying idea of deep learning is to train brain-like networks to perform human tasks (without human intervention). In computer vision, for example, tasks of interest are image classification, object detection, object segmentation, etc.
There are different ways to train the networks: supervised learning, unsupervised learning, and semi-supervised learning. The supervised manner uses a labeled dataset to train a network, allowing it to compare its predictions to the true

labels. In contrast, unsupervised learning includes methods that do not use any labels. Semi-supervised learning is the bridge between supervised and unsupervised learning. It takes advantage of both an unlabeled part of the dataset and a labeled one.

Nowadays, the most studied and developed methods are the supervised ones. However, they suffer from some limitations regarding the dataset it requires to obtain good results. Not only do they need a fully annotated dataset, but also a large one. For a field like the medical one, this is a significant limitation. Indeed, only medical experts can provide accurate annotation of medical images, and it is time-consuming. Hence, to overcome this limitation, unsupervised and semi-supervised methods are increasingly in the spotlight.

From then on, this master thesis also focuses on different ways to overcome this problem. Two main aspects are studied:

- How to complete a partially annotated dataset,
- How to train networks to learn with few annotated data.

The first aspect focuses on the use of deep networks to enrich a dataset, hence alleviating the burden of the annotation process. More precisely, this aspect aims to automatically complete the annotation of a dataset following a training process on an annotated part of it. The second one studies contrastive learning, a recent unsupervised pre-training method that has proven its efficiency by outperforming supervised methods on images from everyday life. Both aspects join each other to perform the same final task, the classification of images containing pulmonary embolisms or not.

This thesis is organized as follows: the second chapter is dedicated to a brief review of deep learning for image classification and its main theoretical aspects. Then, contrastive learning is presented in the third chapter. This last one includes a general introduction to a subset of unsupervised methods and a more detailed presentation of the specific method implemented in this work. Chapters 4 and 5 are more technical as they contain all the implementation details of this thesis. Chapter 4 focuses on the annotation enrichment part, while Chapter 5 presents the general image classification. Finally, before concluding this work with Chapter 7, the results of these two tasks are reported in Chapter 6. This last includes supervised and contrastive results for each task, as well as some areas for improvement regarding the latter. Then it discusses the current place of deep learning in pulmonary embolism detection and the perspective of future work in line with this thesis.

CHAPTER 2

# Deep learning for image classification

One of the well-known tasks in computer vision is image classification. This task aims to train a network to assign the right label (among a set of categories) to images. Classifying cats and dogs is an example of such a task.
Image classification has been widely used in the medical field for different purposes, such as the classification of skin and breast cancers, lung nodules, etc.

Nowadays, deep learning has outperformed previous computer vision methods for this task. The most used networks to do so are **Convolutional Neural Networks** (CNNs), which will be further detailed in Section 2.1.
Indeed, CNNs reached state-of-the-art results, especially since 2012 when AlexNet won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [13].
After that, those networks quickly became a hot topic. In only three years, several improved CNNs have been developed, such as VGG [14], GoogLeNet (also named Inception) [15], **Residual Network** (ResNet) [16], etc. All of these networks became the winners of the ILSVRC they competed in. They achieved better results (shown in Table 2.1) year after year until 2016. They even reached an error smaller than the human one in 2015 with ResNet [12].

After achieving such convincing results in this domain through the ILSVRC, the focus moved to other branches of deep learning. Eventually, although some other networks have been developed after 2015 (sometimes from combinations of the ones mentioned above), ResNet stays still now a typical and efficient CNN to use for image classification. Therefore, this architecture has been chosen as a point of comparison throughout this thesis.

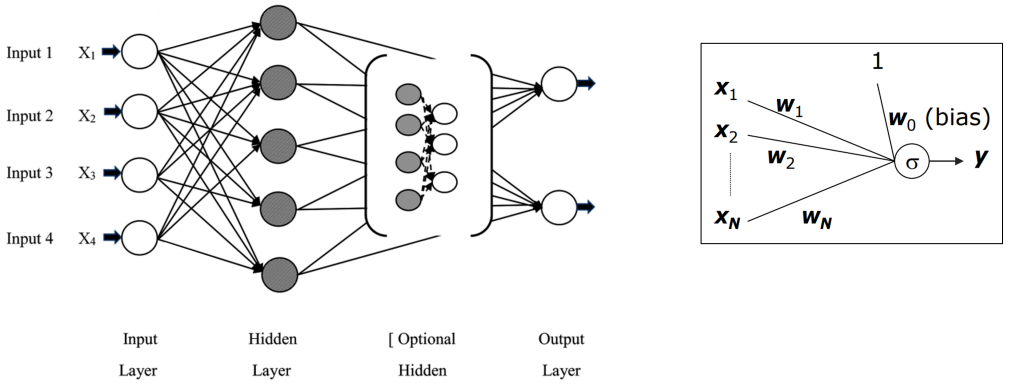
| Features | AlexNet | VGG | Inception-v1 | ResNet | Inception-v2 | Inception-v4 |
|---|---|---|---|---|---|---|
| **1st release year** | 2012 | 2014 | 2014 | 2015 | 2015 | 2016 |
| **Top-5 error** | 16.4% | 7.3% | 6.7% | 3.57% | 4.8% | 3.08% |

Table 2.1: Results of typical CNNs in the ImageNet challenge over the past years [17].

## 2.1 Convolutional Neural Network

CNN is one of the most popular deep neural networks. It has been a groundbreaking discovery, allowing to significantly reduce the computational complexity and the number of parameters of deep networks. Indeed, compared to the Artificial Neural Networks (ANNs), some new layers have been added to serve these purposes. [18].

ANNs are a subset of machine learning in which brain-like networks are developed to handle tasks like pattern recognition that includes classification and prediction problems [19]. The elementary ANN network is visible in Figure 2.1a. It is composed of an input layer, one or several hidden layers and an output layer. Those are, **fully connected layers**. In a fully connected (also called dense) layer, each node (also called artificial neuron or processing unit) of a layer is connected via a weighted link to all nodes of the previous and following layers.




(a) Basic ANN architecture [19]. (b) Artificial neuron [20].

Figure 2.1: Artificial Neural Network.

Figure 2.1b illustrates the detailed functioning of what happens for each unit, and

the mathematical expression of the output of one unit is the following:

$$\mathbf{y} = \sigma(\mathbf{x}^T\mathbf{w} + w_0). \tag{2.1}$$

In the above, $\mathbf{y}$ is the output vector, $\mathbf{x}$ the input vector, $\mathbf{w}$ the weight vector, and $\sigma()$ the **activation function**. This last has several purposes. First, it brings non-linearity into the network allowing to model non-linear functions. Second, it bounds the output value, which helps the convergence of the network [21].

Throughout the training of an ANN, the weights are updated to minimize a loss function (i.e., the error between the actual and predicted output values). The update is managed by a back-propagation algorithm [22]. Indeed, a complete cycle through the network (i.e., an iteration) is composed of two stages [23]: a **forward pass** during which each unit computes its output according to Equation 2.1, and a **backward pass** during which the weights are updated according to the error. The repetition of this scheme constitutes the learning process of a neural network.

ANNs however, are limited in several ways [24]. First, by using only fully connected layers, ANNs are no longer appropriate when the number of input units increases (for example, when inputs are images or volumes). Indeed, the number of parameters to manage increases, which leads to a loss of efficiency as those networks become deeper. Second, fully connected layers are sensible to spatial changes, whereas, for instance, for object detection problems, the network should be able to detect the same object in different locations.
Therefore, CNNs (Figure 2.2) have been introduced to overcome these problems.

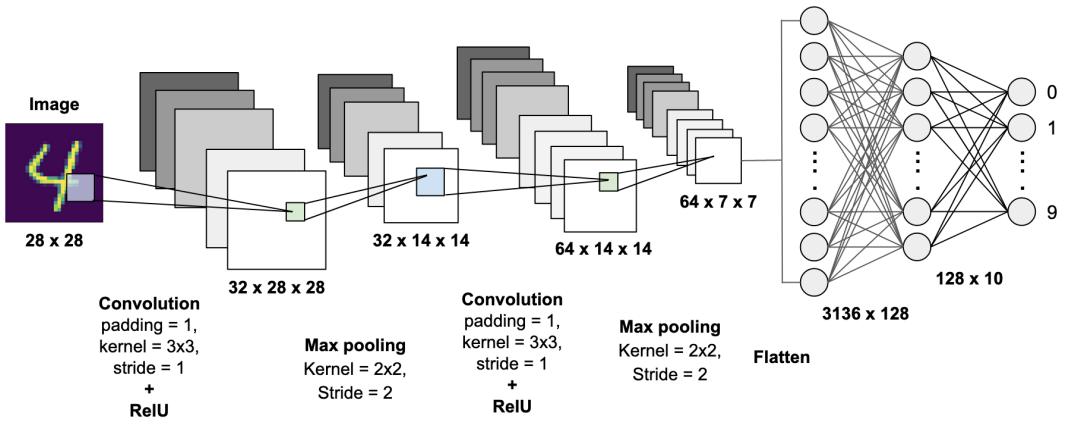



Figure 2.2: Basic CNN architecture.

CNNs have specific layers, such as **convolutional layers** and **pooling layers**. These allow significantly reducing the number of parameters. As shown in Figure

2.2, the first part of the overall architecture, known as the feature extractor, is an alternation of those layers. As for the second part (i.e., the classification head), it is composed of fully connected layers.

### 2.1.1 Convolutional layer

Instead of having each processing unit connected to all the following ones like in fully connected layers, convolutional layers use smaller kernels that move along the input layer. The output is the convolution between one of these kernels (also called convolutional neurons or filters) and all the sections of the input vector, covered one after the other by the kernel while it moves (Figure 2.3).

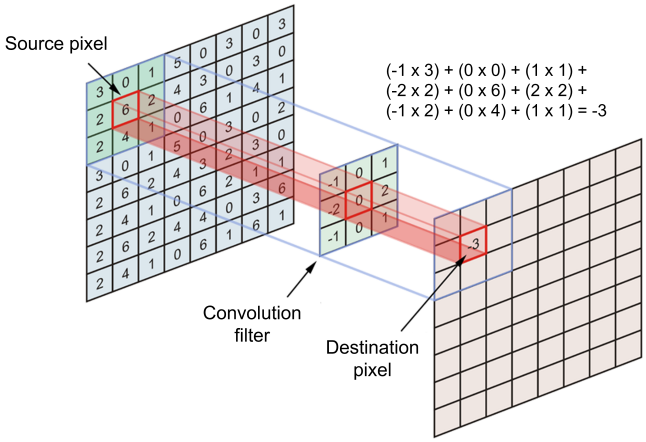
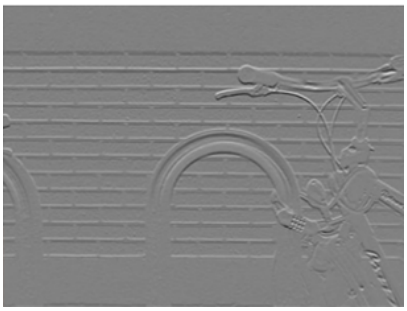



Figure 2.3: Convolution operation between two layers [25].

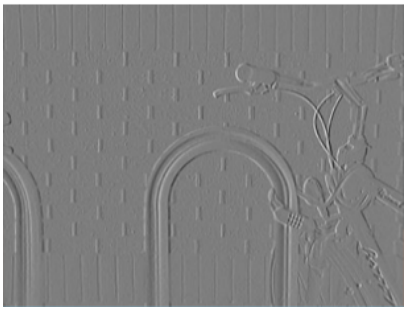
This results in one feature map per kernel, visible in Figure 2.2 (represented by the squares in the depth of each layer). Each map contains different characteristics (for instance, horizontal and vertical edges, corners, etc.) of the image depending on the kernel values. The deeper the kernel in the network, the more complex the extracted features, which ultimately makes it possible to recognise, for example, objects in images. Figure 2.4 illustrates an example of two feature maps obtained by the convolution of two different kernels with the original image.

Another advantage of convolutional layers is that the kernels do not change while sliding along the input. This property is called **weights sharing** [27]. Indeed, kernels keep the same weights throughout the convolutions they perform. In other words, the number of trainable weights corresponds to the dimension of the filter, which drastically reduces the overall number of parameters (see Figure 2.5).

(a) Original image.



(b) Resulting feature map for a kernel that extracts horizontal edges.



(c) Resulting feature map for a kernel that extracts vertical edges.

Figure 2.4: Effects of convolution with two different kernels [26].

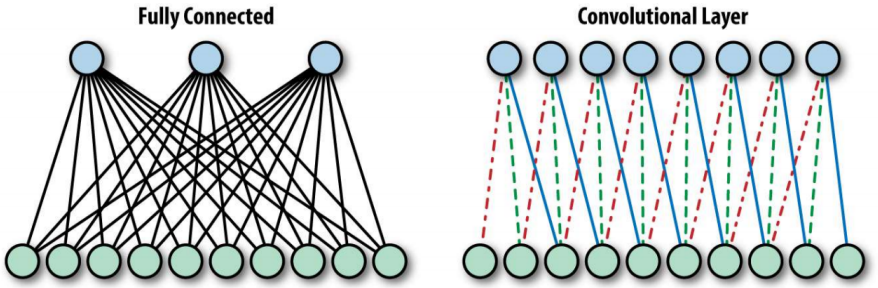



Figure 2.5: Difference between fully connected and convolutional layers [28].

In the fully connected case, for an input layer of ten neurons and an output one of three, there are 30 different weights. As for the convolutional case, for the same input layer and one kernel of size 3, there are only three different weights. This example already shows a significant reduction in parameters while the output layer of the fully connected case contains fewer neurons.

Finally, convolutional layers bring a last interesting property known as **shift-invariance**. Indeed, if the input slightly shifts, the corresponding feature map will shift in the same way without being otherwise modified [29, 27].

### 2.1.2 Pooling layer

The other kind of layer used to reduce the computational complexity of deep networks is the pooling layer. The latter allows reducing the spatial dimension of the feature maps while adding no parameters to the network. The best-known pooling layers are the **max pooling** and **average pooling** layers. The way they operate is quite similar.

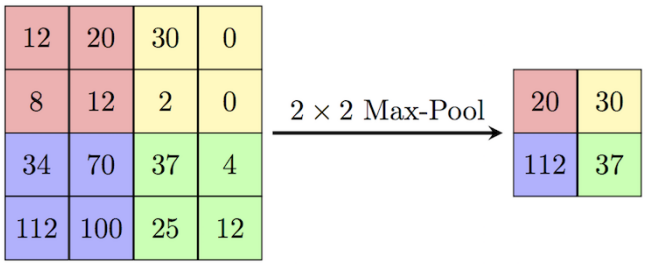



Figure 2.6: Example of max pooling operation of filter size and stride of 2 [25].

The max pooling operation, visible in Figure 2.6, reduces the dimension of its input by keeping only the maximum value of each section of the input covered by a filter that slides along it. The size of the filter and the stride (by how much the filter moves) are chosen.
The average pooling, as its name suggests, keeps the average value of the section covered by the filter instead of the maximum value.

## 2.2 Model architecture

As said previously, the network implemented in this thesis is a ResNet. Its particularity is that **residual blocks** have been added to the basic architecture of CNNs to avoid accuracy degradation problems when increasing the network depth [16]. While increasing the depth of a network might provide better results, above a certain point, it can cause degradation due to the vanishing gradient issue [30]. Therefore, the motivation of ResNet is that deeper networks could perform as well as their shallower counterparts if the extra layers could learn the identity function and copy the results of those shallow networks.

However, the identity function is not that easy to learn for a network, hence the usefulness of residual block, illustrated in Figure 2.7.
This configuration allows the model to learn identity mapping more easily. Indeed, it is easier for weight layers to converge to a zero function than the identity one.

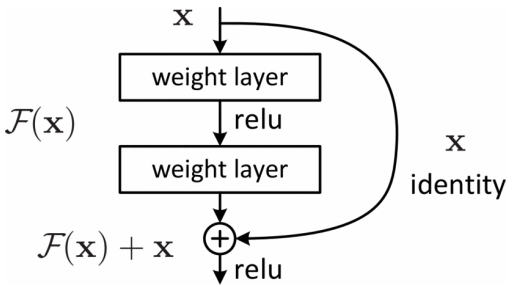



Figure 2.7: Residual learning: A building block [16].

The above motivation has led to the ResNet hypothesis: «it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping »[16].

There are different ResNet networks according to their depth. In this thesis, the most common one, the ResNet50, is used. Figure 2.8 shows its architecture.

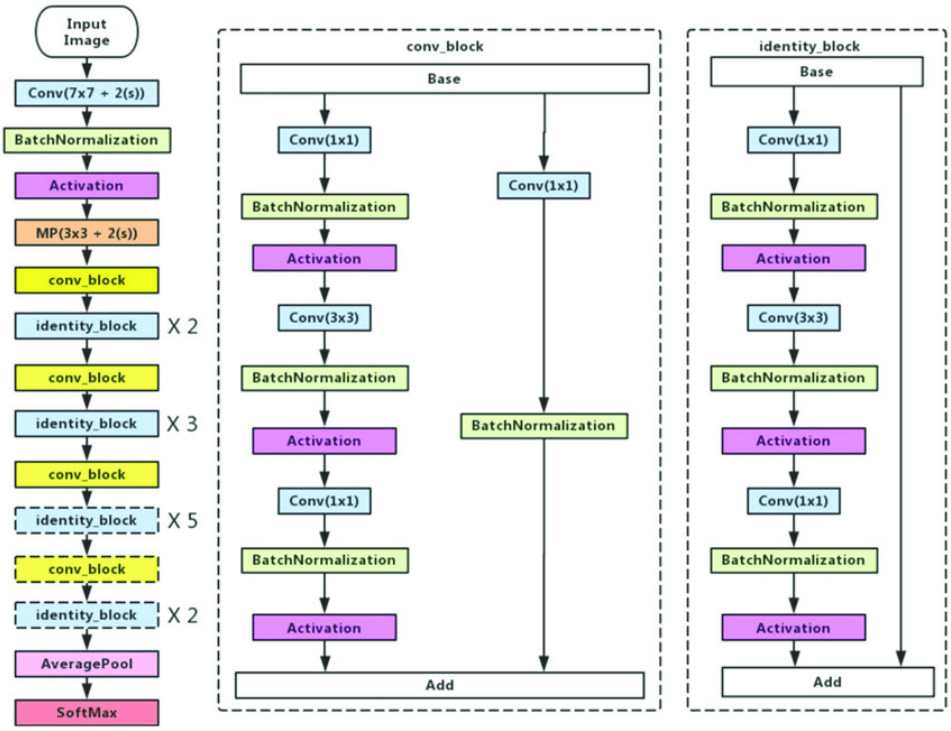



Figure 2.8: Architecture of ResNet50 with the detailed implementation of the residual blocks [31]. *Conv* stands for convolutional layer and *MP* stands for max pooling layer.

The network contains two different residual blocks: identity block and convolutional block. However, the last one is only used when the input and output dimensions of the block are different. Otherwise, the identity block is preferred as it does not add extra parameters or computational complexity [16].
Concerning the activation function, a **Rectified Linear Unit** (ReLU) function [32] is used.

## 2.3 Supervised learning issue

Up to now, supervised methods have been the most used. However, although they have proven to be efficient methods, they raise a considerable problem, especially for the medical world. Those methods need accurately annotated large datasets to train and achieve good performance. However, those are not that easily built because acquiring relevant annotations is time- and people-consuming as it can only be done by medical experts. Therefore, this hinders the progress of deep learning in the areas that could benefit the most.

This issue, which is the concern of this thesis, has pushed the researchers to focus on different areas of deep learning. In this work, two tasks have been implemented with two different methods (see Figure 2.9), one supervised and the other based on an unsupervised pre-training. This last one, further described in Chapter 3, allows networks to learn good image representations without annotations. Thanks to this pre-training (called contrastive learning), a network is supposed to reach the same or even better level of performance than its supervised counterpart when decreasing the number of annotated data.

| Approaches / Methods | Annotation enrichment | General classification of medical images |
|---|---|---|
| **Supervised learning** | *Case1* | *Case2* |
| **Contrastive learning** | *Case3* | *Case4* |

Figure 2.9: Cases studied throughout this master thesis.

Figure 2.9 lists the four cases that have been developed throughout this thesis. These allow comparing both methods. In addition, they point out that there is not always the need for a new technique to address the issue. Indeed, another task

performed with the same method (hence suffering from the same limitations) can mitigate the problem.

The first task (Case1 and Case3) is already part of a solution concerning the supervised issue. It consists of training a network to complete partially annotated datasets. Therefore, this task can help to create fully annotated datasets while decreasing the workload of medical experts.
The second one (Case2 and Case4) keeps the usual training purpose of creating a model able to generalize its performance (in this context, the classification of images with and without pulmonary embolism) to other images besides the ones used for the training.

CHAPTER 3

# Unsupervised pre-training and fine-tuning

Unsupervised representation learning (i.e., unsupervised pre-training) aims to extract meaningful and relevant features (also called representations) from unlabeled images to optimize the network performance for a downstream task [33].

Currently, this is a hot topic of deep learning as it opens a new field of possibilities for learning processes. Indeed, even though the downstream task still requires a labeled dataset, experiments [34] show that networks that undergo these unsupervised pre-training steps outperform those that have learned from scratch in a supervised manner. Moreover, with the pre-training, networks also seem to achieve better performance as the fraction of labeled data decreases. Therefore, this could mitigate the primary problem of deep supervised learning (see Section 2.3) while outperforming it.

Self-supervised methods are considered a subset of unsupervised representation learning. Although unsupervised, they are still quite close to supervised methods. They are similar in the sense that they both use labels to train their networks. Nevertheless, self-supervised methods create pseudo-labels from the input data, and their training phase is usually done by accomplishing some pretext tasks [35, 36]. This way, they can extract relevant image representations for downstream tasks.

Self-supervised methods can be classified into different groups according to their final task. Previously, most of those methods were **generative** ones. Those aim to create real-looking images through tasks such as image colorization [37].
Although those methods efficiently achieve their goal, they can be computation-

ally expensive. Moreover, they are not the most adapted to learn relevant image representations [38]. Recently, other methods called **discriminative** have shown remarkable results [39]. In those methods, networks perform a pretext task (such as rotation prediction [40], solving jigsaw puzzles [41], etc.) to learn the best representations possible from an unlabeled dataset.

One groundbreaking technique, part of the discriminative ones, has recently been developed: **contrastive learning**. It has already achieved promising results on image classification with, among others, the Imagenet dataset [42]. In the context of this thesis, one contrastive method has been implemented to compare the results to those obtained in a supervised way.

The following chapter is divided as follows: the first section addresses the theoretical basis of contrastive learning. This leads to the presentation of the specific model implemented in this thesis: **SimCLR** [43]. Finally, the last section describes the process of transferring a pre-trained network for a downstream task.

## 3.1 Contrastive learning

The idea behind contrastive learning is to learn meaningful representations by comparing input samples together. The purpose is to differentiate similar elements (i.e., positive pairs of inputs) from dissimilar ones (i.e., negative pairs). These methods use the notion of distance in the representation space (space of lower dimension than the input one, containing the extracted features, see Figure 3.1) to achieve this goal. Indeed, they aim to link the notions of similarity and distance. For this purpose, representations of similar inputs are clustered in the representation space while being pushed away from representations of dissimilar ones. Then, as said previously, the extracted features are reused for a downstream task via a transfer of the pre-trained network.

Figure 3.1 shows the overall pipeline of contrastive learning.

The first step of this process is to create those positive and negative pairs. A common and quite simple practice in computer vision is to use data augmentation[1]. In that case, each image from the batch[2] is augmented twice to create a positive pair.

[1]Data augmentation consists of applying transformations to images such as rotation, shift, cropping, etc.

[2]A batch is a sample of the dataset that contains the images that are used during an iteration.

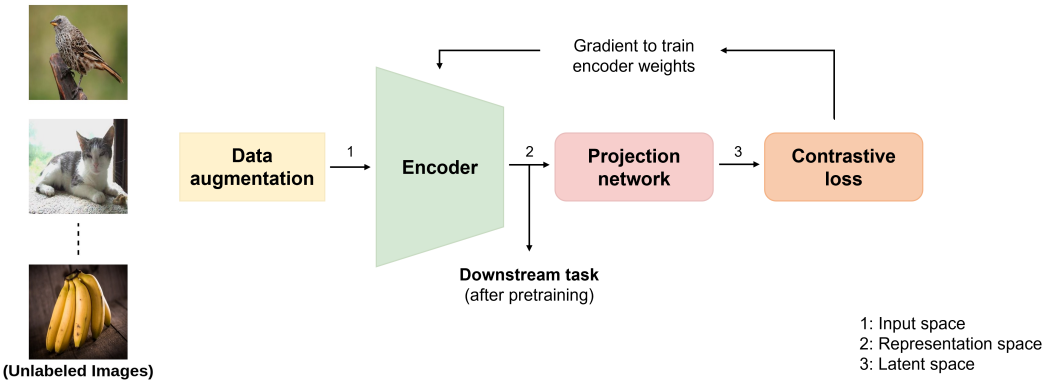



Figure 3.1: Pipeline for contrastive learning.

Any pair formed by one of those two images with another one is therefore considered a negative pair.
Then, the batch of augmented images goes through an encoder network to extract its representations.
Next, the representation vector (i.e., the output of the last layer of the encoder) becomes the input of an extra projection network that projects it to the latent space (a lower dimension space containing the data in a compressed manner).
Finally, the resulting vectors are used to compute the contrastive loss that serves to update the network weights.

### 3.1.1 Mechanisms

Several contrastive mechanisms have already been developed. They differ in two points: how networks get access to negative samples during the training and how the encoder is updated. Those mechanisms are illustrated in Figure 3.2. *The following explanations are mostly based on these two sources [36, 39]*

The first mechanism shown in Figure 3.2a represents the **end-to-end learning** implemented among others in SimCLR (Simple framework for Contrastive Learning of visual Representation) [43] and CPC (Contrastive Predicting Coding) [45]. In this mechanism, contrastive learning benefits from many negative samples. Therefore, this method uses large batches to provide a sufficient number of negative samples. Moreover, both encoders can be different and are updated directly using backpropagation.
However, the number of negative samples is directly linked to the batch size, which is the major limitation of those methods. Indeed, as said previously, contrastive

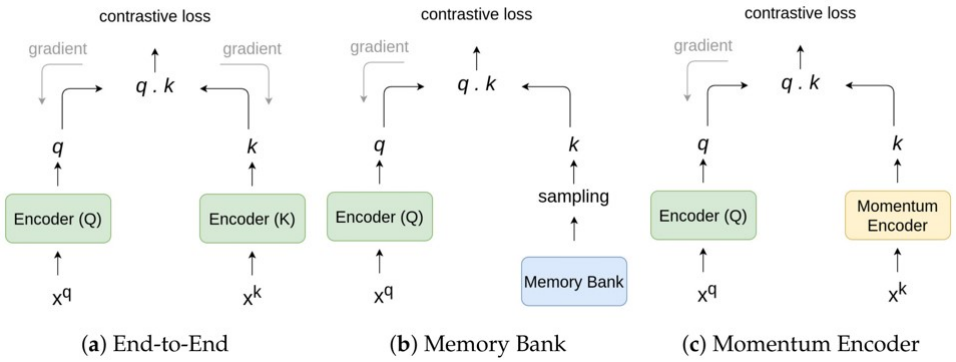



Figure 3.2: Pipelines for contrastive learning mechanisms. Illustration for a pair of query($x^q$) and key($x^k$)[1] [36, 44].

learning benefits from many negative samples, but the batch size is limited by the GPU memory size hence limiting the model performance.

The second mechanism (Figure 3.2b), implemented in multiple methods [46, 47, 48], allows alleviating the previous problem by storing the representations (computed in previous epochs[2] by the query encoder) for each image in the dataset into a separate **memory bank**. Therefore, the negative samples can be sampled from the memory bank, allowing to train with many negative pairs without significantly increasing the computational costs.
However, updating the representations kept in the memory bank can be computationally expensive and a complicated task (to keep them consistent).

The third mechanism [44] (Figure 3.2c) also addresses the issues from the previous methods, especially those arising from the memory bank. In this mechanism, a momentum encoder that shares the same parameters as the encoder replaces the memory bank. As opposed to the end-to-end learning, the momentum encoder is not updated with back-propagation but based on the query encoders parameters. Therefore, only one of the encoders needs to be trained with this method. Figure 3.3 shows the other particularity of the momentum encoder. It uses a dictionary of keys as a dynamic memory queue (instead of a memory bank). This dictionary is updated at each iteration (compared to each epoch for the memory bank).

[1]Query and key refer to particular views of an input sample, forming either a positive or negative pair according to the image from which they are derivated [39].

[2]An epoch consists of a set of iterations. It is completed when all the images have been passed through the network once.

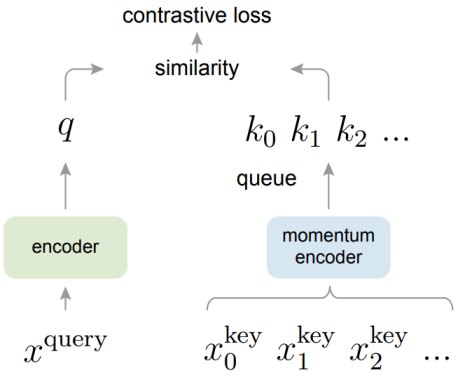



Figure 3.3: Momentum Contrast pipeline [44].

## 3.2 SimCLR

SimCLR [43] is one well-known successful example of end-to-end learning. Indeed, it has proven to be efficient while being trained on ImageNet [42]. Moreover, its simplicity makes it a good starting point for the first implementation of a contrastive method.
Therefore, this section reviews the pipeline described in Section 3.1 for this specific model implemented for this thesis.

### Data Augmentation

Data augmentation is an essential step of the SimCLR pipeline. Indeed, according to the applied data augmentation, the results can be significantly affected. Therefore, after studying the impact of several data augmentations (alone or by combining several), SimCLR found a winning combination. For this model, they use random cropping and resizing, color distortion, and Gaussian blur to create augmented views of images.

### Encoder network

Unlike the choice of the data augmentation technique, the choice of the encoder network is almost free. Indeed, contrastive learning results are less dependent on the encoder architecture hence allowing various possibilities. However, SimCLR shows that contrastive learning benefits from deep and wide networks by performing several tests with different encoders. To do so, they chose to use the common ResNet architectures as encoders.

### Projection head

Although practically similar to the encoder in the sense that they are both parts of the network, the projection head has a different purpose. Indeed, while the encoder extracts representations, the head acts as a bridge decreasing the dimension of vectors between the representation space and the latent space.
After training, the projection head is discarded to keep the best representation (i.e., the output of the last layer of the encoder) for the downstream tasks. Therefore, the head allows efficient computation of the similarity metric (i.e., contrastive loss) without either suffering from significant computational costs or losing information in the representation space due to a premature dimension reduction.
In SimCLR, several projection heads have been tested to study their impact and importance. They finally opted for a non-linear projection head composed of two fully connected layers separated by a non-linear activation function (ReLU).

### Contrastive loss

Just as there are several possible contrastive architectures, different contrastive loss functions are used [39]. However, the most common one is **InfoNCE** [45] based on NCE (Noise-Contrastive Estimation) [49].
The idea behind InfoNCE is motivated by the supervised softmax classification problem. This one, defined by Equation 3.1, tends to maximize the probability of classifying a sample in the correct class.

$$p(i|\mathbf{q}) = \frac{\exp(\mathbf{q}^T \mathbf{w}_i)}{\sum_{j=1}^{n} \exp(\mathbf{q}^T \mathbf{w}_j)}\,, \tag{3.1}$$

where $\mathbf{q}$ is the feature vector and $\mathbf{w}_{i/j}$ the weight vectors of class i/j.

The above expression can be adapted to the contrastive approach, which consists in comparing two samples. Therefore, the probability to be maximized becomes the probability of correctly identifying a positive pair among a set of negative ones [39, 46]. Equation 3.1 becomes

$$p(\mathbf{k}^+|\mathbf{q}) = \frac{\exp(\mathbf{q}^T \mathbf{k}^+)}{\sum_{k \in \mathcal{K}} \exp(\mathbf{q}^T \mathbf{k})}\,. \tag{3.2}$$

$\mathcal{K}$ is the set of keys. $\mathbf{k}^+$ is the key that forms a positive pair with the query $\mathbf{q}$ while $\mathbf{k}$ forms a negative pair.

The issue with Equation 3.2 is that the denominator can be problematic to calculate [49], particularly when the size of $\mathcal{K}$ increases. Therefore, NCE provides a way to estimate this probability by evaluating this denominator through a binary

classification between the data and some artificial noise [49]. This principle has then been extended in [50] to a global version. This last is not suited for a binary task anymore but for the ranking of a set of keys according to their similarity to a query [39]. The probability function becomes therefore

$$p(i|\mathbf{q}, \mathcal{K}) = \frac{\exp(S(\mathbf{q}, \mathbf{k}^+))}{\sum_{k \in \mathcal{K}} \exp(S(\mathbf{q}, \mathbf{k}))}\,, \tag{3.3}$$

where $S(\mathbf{q}, \mathbf{k})$ is the function that calculates the similarity between $\mathbf{q}$ and $\mathbf{k}$.

Finally, since the objective is to maximize the above probability, the equivalent problem becomes the minimization of Equation 3.4.

$$\mathcal{L}(\mathbf{q}, \mathcal{K}) = -\log p(i|\mathbf{q}, \mathcal{K})\,. \tag{3.4}$$

SimCLR uses a variant of this loss they name **NT-Xent**, which means Normalized Temperature Cross-entropy loss. In this loss function, the similarity metric is the **cosine similarity**, expressed as follows:

$$sim(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^T \mathbf{v}}{||\mathbf{u}||\,||\mathbf{v}||}. \tag{3.5}$$

The loss function between two representation vectors $\mathbf{z}_i$ (i.e., the query) and $\mathbf{z}_j$ (i.e., the positive key) forming a positive pair is then defined as

$$l_{i,j} = -\log\left(\frac{\exp(sim(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]}\, \exp(sim(\mathbf{z}_i, \mathbf{z}_k)/\tau)}\right). \tag{3.6}$$

In the above, $\tau$ is a temperature coefficient that has an impact on the strength of penalties given to the hard negative samples (i.e., the nearest neighbors of the query [51]), $2N$ is the size of the batch that is composed of two augmented versions of $N$ input images, $\mathbf{z}_k$ is a representation vector that forms a negative pair with $\mathbf{z}_i$, and $\mathbb{I}_{[k \neq i]}$ is a function that equals 1 if and only if $k \neq i$ otherwise it is null.

**NB**: Methods based on infoNCE loss can lead to a trivial solution and need to be carefully implemented. Therefore, to avoid this collapsing possibility, Barlow Twins [52] implements a loss function that makes the cross-correlation matrix between two representation vectors tend to the identity matrix. In addition to avoiding trivial solutions, this method does not require large batches, which alleviates the other issue of contrasting methods.

## 3.3 Downstream task

The pre-trained model is reused in a downstream task to evaluate the efficiency of self-supervised learning. The hypothesis is that if the model has learned good representations, it can efficiently perform while being transferred for the final task.

This evaluation of representations can be done in several ways: supervised learning, **semi-supervised learning**, or **transfer learning**.
In the case of supervised learning, the dataset used for contrastive learning must be fully annotated. Therefore, the downstream task can also use it. In that case, contrastive pre-training would be used to try to outperform supervised training from scratch.
Semi-supervised learning is the bridge between supervised and unsupervised learning. The training of a semi-supervised method is done with a partially annotated dataset, hence taking advantage of both a dominant unlabeled part and a smaller labeled one. In this case, the whole dataset is used for the contrastive learning while only the small annotated part is used for the downstream task. Given the results already obtained by researches on the subject, the semi-supervised way allows mitigating the supervised limitation while still achieving convincing results (at least better than the supervised ones) on a limited set of annotated data.
Finally, transfer learning aims to perform the downstream task with the pre-trained network but on another dataset than the one used for the pre-training. The knowledge from this last one is thus transferred to another dataset, hoping that the network had previously learned good general representations. This approach is especially used when the available dataset is too small. However, the outcome strongly depends on the similitude between both datasets.

For the sake of clarity, the two first manners will be part of a general process that will now be referred to as *network reuse*. This is to ensure that there is no misinterpretation between the transfer of the network to another dataset (transfer learning) and the reuse of the network on the same dataset (in a supervised or semi-supervised way).

### 3.3.1 Fine-tuning

The concept behind the process of taking a pre-trained model as a basis to perform another task (via transfer learning or network reuse) is called fine-tuning. Indeed, instead of starting from scratch with randomly initialized weights, the model already has pre-trained weights. The latter will therefore be fine-tuned during the downstream task provided they are still updated during it. By doing so, the

downstream task benefits from the previous knowledge of the pre-trained network.

The main advantage of fine-tuning is that it allows the model to reach satisfying results even with small datasets. Indeed, the main issue with small datasets is overfitting. This means learning the training set so well that the model cannot generalize what it has learned on another dataset. However, as the fine-tuning process starts with a pre-trained model, which has already learned relevant features, the risk of overfitting is reduced [53].

As explained in Section 2.1, CNNs are composed of two distinct parts, the backbone of the network known as feature extractor and the head designed for a specific task. The only part that needs to be changed while fine-tuning a network for another task is the head.
In this case, after the contrastive learning, the non-linear head of the network is discarded to keep only the encoder whose last layer contains the learned representations. Therefore, a new classification head is added to the encoder then the whole network is trained to perform the final binary classification.
Moreover, the encoder weights are already supposed to be optimized, provided the extracted representations are relevant. Therefore, a common practice when reusing or transferring the network consists to first freeze those weights to only train the new head until convergence. Then the fine-tuning can be done by unfreezing all the weights.

CHAPTER 4

# Annotation enrichment

As said in Section 2.3, the first studied task of this thesis is the enrichment of annotation. This task aims to complete a partially annotated dataset to make it usable for supervised learning. Therefore, it alleviates the annotation burden even though it does not solve the supervised learning issue.
In practice, the network is trained several times with fewer and fewer labeled data. Then, for each training, the network must give its predictions about the remaining unlabeled data. The objective of this work is to quantify the extent to which it is possible to reduce the required amount of labels.

This chapter contains all the details about the implementation of this task. It goes through the dataset presentation, the image preprocessing and augmentation, the value of the hyperparameters, and ends with the loss, optimizer, and architectural choices. Moreover, as also mentioned in Section 2.3, this task has been implemented with two methods. Therefore, both supervised (Case1) and contrastive (Case3) implementations are discussed in this chapter.

## 4.1 Supervised implementation: Case1

### 4.1.1 Dataset

The networks have been trained on the public RSNA (Radiological Society of North America) Pulmonary Embolism CT Dataset [6]. It consists of 2 995 147 images from 12 195 patients with 96 540 (4.2%) images containing a form of pulmonary embolism. These will now be referred to as *positive images/cases*. It is a strongly **imbalanced dataset** which is an issue faced a lot in the medical field.

The images are obtained by Computed Tomography Pulmonary Angiography (CTPA), which is the most current imaging modality in the diagnosis of pulmonary embolisms [54, 55].
The dataset is fully annotated at image and study levels. However, in this work, the focus is on the image level annotation, that is, if there is any form of pulmonary embolism present in the image or not. This results in a binary classification.

From this dataset, a subset of 15556 images, containing 4.21% of positive images (i.e., 655 images), is used. For the sake of clarity, this subset will now be referred to as *Dataset1*.
Dataset1 is divided in three parts known as **training set**, **validation set**, **test set**. The training and validation sets are used during the training process of the network. The training set consists of the data used to train the model and update the weights. It is the set from which the model will learn. The validation set contains the data used to evaluate the model performance throughout the training. It provides other images to the model to have an unbiased evaluation of its learning. Finally, the test set is kept for the last performance tests once the model is trained.

As this case aims to complete partially labeled datasets, images intended for the training are sampled from Dataset1. That is, by removing images from Dataset1 for several sampling rates.
For instance, if the sampling rate is 9/10, nine images out of ten are kept for the training stage. Then, this part of Dataset1 is randomly split between the training set (80%) and the validation set (20%). The remaining 10% (i.e., one image out of ten) constitutes the test set, as illustrated in Figure 4.1.

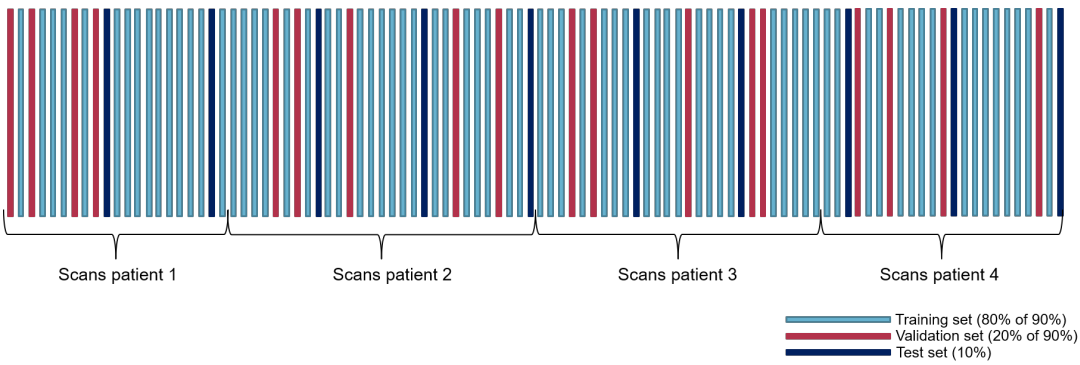



Figure 4.1: Separation of dataset into training, validation and test sets in Case1.

This way of separating Dataset1 is specially chosen for this task as images from the same patients are used during the training and testing processes. The underlying

hypothesis of this separation is that training the network with part of images from a patient allows it to complete the annotation of the remaining scans of this same patient. Therefore, it is necessary to have a well-defined sampling rate and not to separate Dataset1 randomly.

### Preprocessing

All the images undergo preprocessing steps before entering the network.

Firstly, as the format of the CTPA images is DICOM (Digital Imaging and Communications in Medicine), they have to be converted. In CT scans, the density of matters is expressed in Hounsfield units (HU), and it covers a range of 2000 HU (from the density of air (-1000 HU) to the density of metal (1000 HU)). However, according to the task of interest, only parts of this range are relevant. Therefore, a **windowing** process is applied to the image to transfer only these relevant fractions of the Hounsfield range into the grayscale. According to the window settings (the center/level (WL) and width (WW)), the brightness and contrast of the resulting image are different. These settings aim to isolate the various information contained in the CT scans. Figure 4.2 shows the impact of these settings on the same image.

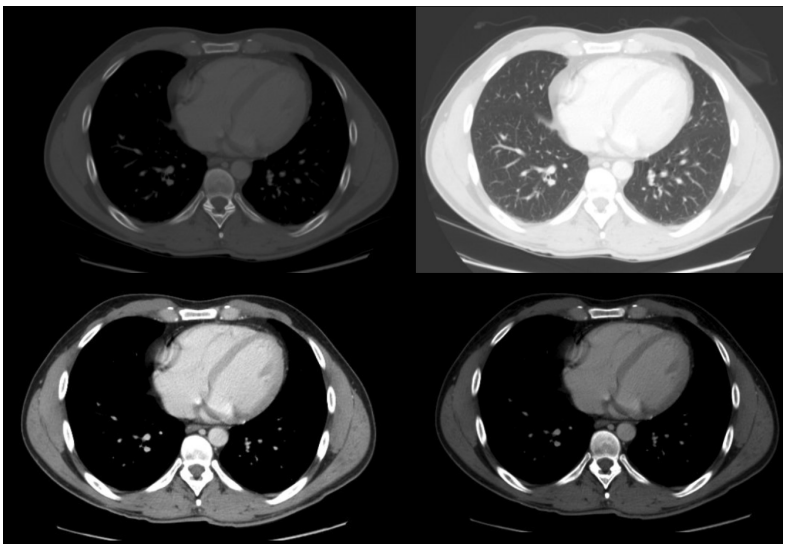
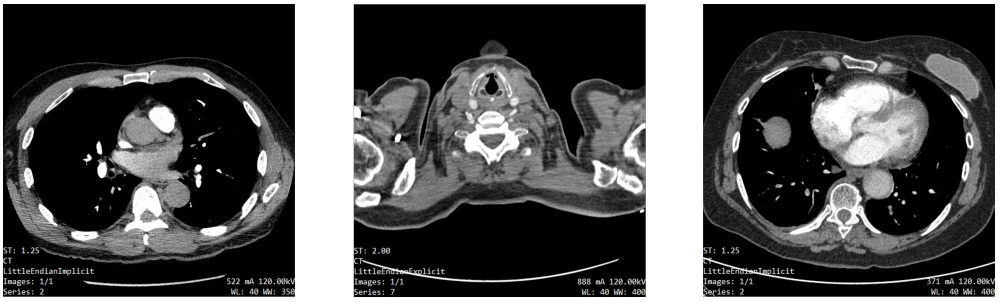


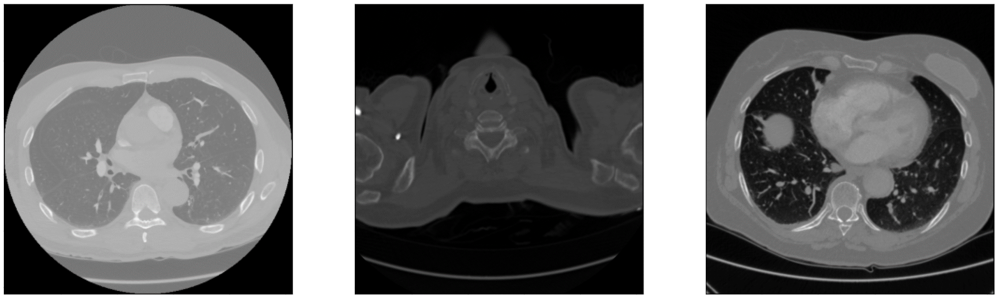
Figure 4.2: Different window settings of the same image [56].

The area of interest of this study being the lungs, three windows have been applied to relate the three dimensions of the RGB color space to relevant ranges of HU. Those three windows focus thus on the lungs (WW=1500, WL=-600), the mediastinum (WW=400, WL=40) (central compartment of the thoracic cavity), and the density level of pulmonary embolisms (WW=700, WL=100) [57]. Eventually, a conversion

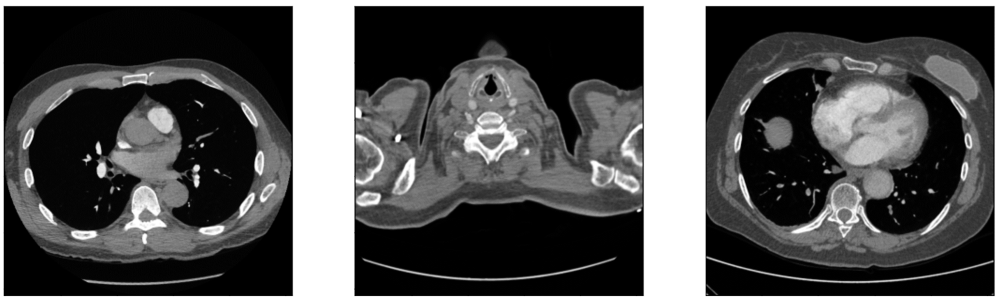
from RGB to grayscale is performed mainly to alleviate the computations. Figure 4.3 shows the result of the windowing process compared to the transfer of the entire range of HU directly into a normalized range of [0,1].



(a) Original CTPA scans.



(b) Resulting images without windowing.



(c) Resulting images after windowing.

Figure 4.3: Difference between loading image processes.

Secondly, the images have been **normalized** to have the pixel value comprised between 0 and 1 and finally **resized** (from 512x512 to 128x128) mainly to save computational time.

### Data augmentation

Classical data augmentation is used throughout the training process. As visible in Figure 4.4, images undergo random data augmentation based on a mix of slight rotation (range of 10°), horizontal and vertical shifts (range of 10% of the image), and zoom (range of 20% of the image).

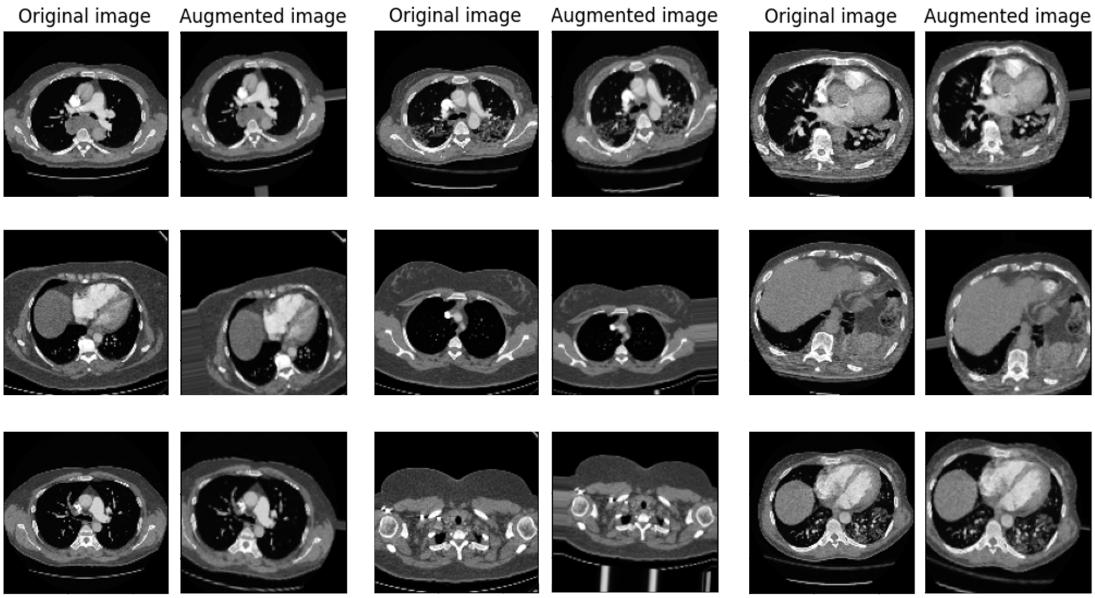



Figure 4.4: Example of random data augmentation the images can undergo.

Data augmentation is also used during the testing phase to measure the uncertainty of the predictions. Indeed, instead of directly providing the test set to the network, each image is augmented four times beforehand. The final labels are obtained by computing the average of the five predictions (the original image and the four augmentations) for each image. This process is called **test-time augmentation** [58, 59].
Although the augmentations are in the range of those used for the training process, as opposed to this one, these four augmentations are not random anymore. Each image undergoes only one augmentation that is either a 10°-rotation, a slight vertical or horizontal shift of 10 pixels (corresponding to 7.8% of the image), or a central crop keeping a central fraction of 75% of the image. Figure 4.5 illustrates these augmentations.

After taking the mean of the five predictions, two thresholds are used to classify the images into two categories: the images whose labeling is reliable and does not have to be rechecked and those for which it is uncertain. If the mean prediction

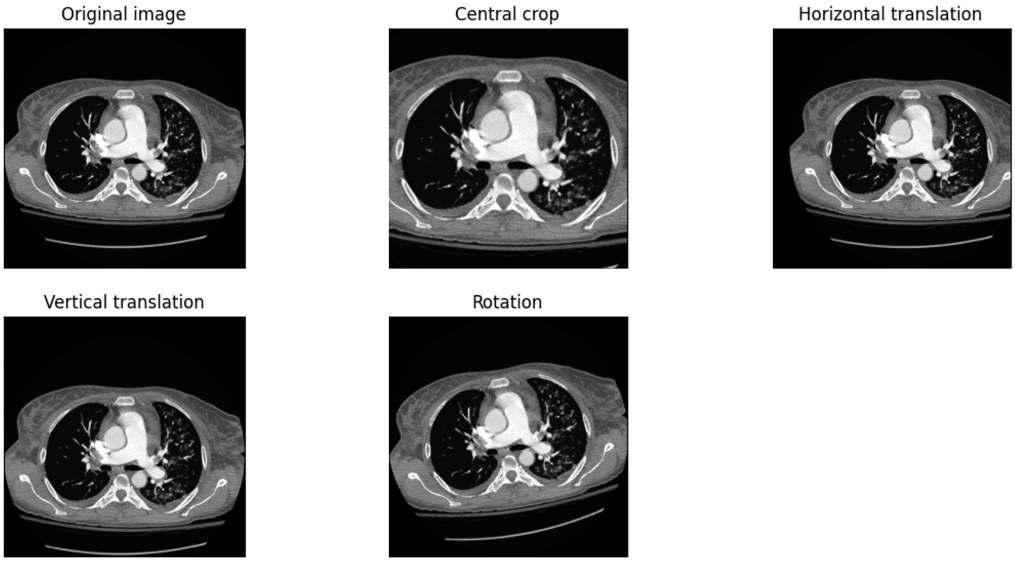



Figure 4.5: Test-time augmentation applied on one image.

is above the upper threshold or under the lower one, the images are considered permanently classified and respectively positive or negative. Therefore, test-time augmentation allows isolating the ambiguity of the model and the images that have to be rechecked by medical experts.

### 4.1.2 Hyperparameters

For training to be efficient, some values of hyperparameters (i.e., parameters that are not trainable) have to be chosen. Although the hyperparameters sometimes have pretty good default values, it is mainly by training the model with different values that the best combinations are found. Indeed, as some good and bad behaviors are associated with both small and large values of those hyperparameters, each experiment must find its good compromise. Moreover, there can be a high correlation between some of them, hence the tuning need for each situation [60]. The following values of hyperparameters have been chosen by trial and error based on common practices.

One of the most important hyperparameters is the **learning rate**, which has a direct impact on network convergence. It controls the step size of the update of the network weights to converge to a minimum loss.

A common practice concerning the learning rate is to decrease its value during the training, that is, using an adaptive learning rate. This allows refining the weights update throughout the process by applying smaller changes to the weights when the network converges. There are different types of adaptive learning rates according to the function they follow. In this thesis, a step function is used with an initial learning rate of 0.01 (Figure 4.6).

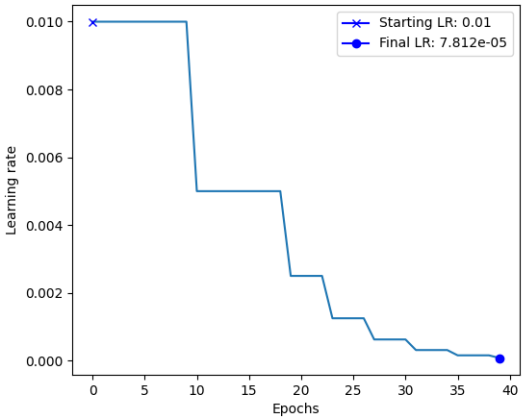



Figure 4.6: Step decayed learning rate.

The **batch size** and the number of **epochs** are other important hyperparameters (others are presented in the subsections afterward).

The batch size is the number of images used during an iteration (i.e., single forward and backward pass in the network). The training set is divided into several batches from which the network is trained, the error is computed, and the weights are updated.

An epoch consists of one complete cycle through the training set. It is an ensemble of iterations and is completed when all the training images are used once. By choosing the number of epochs beforehand, the length of the training process is fixed. Therefore, the number of epochs should be large enough to allow the training to converge but not too large to spare computation time. However (although not used in this case), it is also possible to give a condition based on the performance on the validation set under which the algorithm will stop. This is called EarlyStopping, and it allows to arbitrarily use a large number of epochs without worrying about the computation time.

For this case, the batch size is 32, and the model is trained for 40 or 50 epochs, 50 being for the training with the smallest training and validation sets.

### 4.1.3 Loss function and optimizer

In this implementation, the choice of loss function has been motivated by the imbalanced problem. Therefore, the chosen loss function is the **binary focal loss**, defined by Equation 4.1 [61], specifically designed for this kind of dataset. The

binary focal loss is a generalization of the binary cross-entropy [61] in which an extra hyperparameter (i.e., the focusing parameter) has been added. This one allows the learning process to focus on hard examples, that is, those from the under-represented class. In this work, this hyperparameter has been assigned the value of 2.

$$FL(p_t) = -(1-p_t)^{\gamma}\log(p_t) \quad \text{with} \quad p_t = \begin{cases} p & \text{if} y = 1 \\ 1-p & \text{otherwise.} \end{cases} \tag{4.1}$$

In the above, $y$ is the true label, $p$ is the estimated probability for the class of label $y = 1$, and $\gamma$ is the focusing parameter.

The optimizer is the algorithm used to update the weights of the network to reduce the loss. The optimizer used in this implementation is **Stochastic Gradient Descent** (SGD). Two hyperparameters have an impact on its algorithm. As said previously, the first one is the learning rate. As for the second one, it is the **momentum**. This last one is a term added to the expression of the optimizer that improves the convergence speed [62, 63], especially in the case of convex functions [64].
Figure 4.7a illustrates the impact of the momentum term. In this case, the value of the momentum is 0.9, which is a typical value for this hyperparameter.

(a) SGD without momentum.

(b) SGD with momentum.

Figure 4.7: Impact of the momentum on SGD optimizer [26].

### 4.1.4 Architectural choices

As said in Section 2.1, the trained network is a ResNet50. However, some slight changes have been made compared to Figure 2.8. Indeed, two layers known as **batch normalization** [65] and **dropout** [66] have been added just before the final classification one. Both types of layers are now widely used as they have been shown to have beneficial impacts on the learning process. They both have a regularization effect preventing overfitting. Moreover, batch normalization layers also impact the training speed.

Lastly, the activation function of the last classification layer is not a softmax function as in Figure 2.8 but a **sigmoid** function, well-adapted for binary classification.

## 4.2 Contrastive Implementation: Case3

### 4.2.1 Dataset

The dataset for Case3 is Dataset1. However, it is not divided into several subsets. Indeed, contrastive learning only uses one dataset as its evaluation is done via the downstream task. All preprocessing steps that the images undergo are also the same as described for Case1.

#### Data augmentation

About the data augmentation, as said in Section 3.2, a composition of several random augmentations shown in Figure 4.8 is applied to both samples forming the pair. As in this context, the images are processed in gray-scale, the color distortion initially performed in SimCLR is replaced by contrast distortion only.

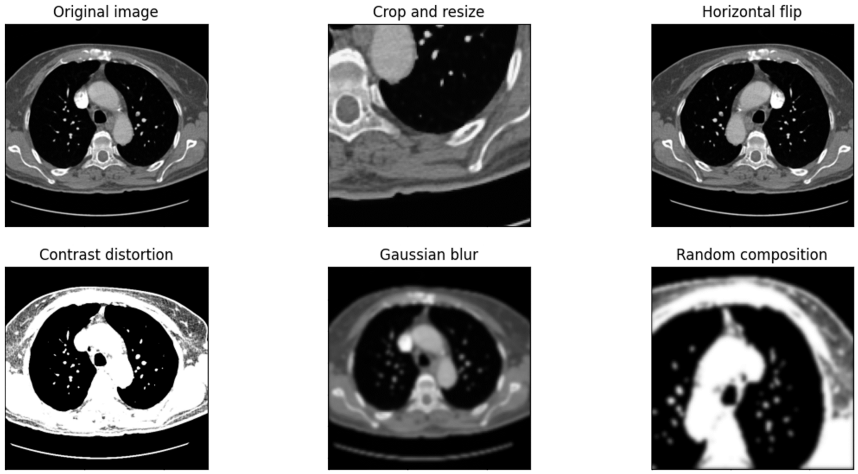



Figure 4.8: Contrastive data augmentation.

### 4.2.2 Hyperparameters and optimizer

Several hyperparameters have a considerable impact on contrastive pre-training. As said previously, SimCLR benefits from large batches to have enough negative samples. Therefore, the chosen batch size is 1000.

The number of epochs is also critical as contrastive learning benefits from long training. Hence, the model is trained once for 75 epochs and then for 150 epochs. Concerning the optimizer, the network is optimized by a stochastic gradient descent algorithm with a momentum of 0.9 and a constant learning rate of 0.01.
Finally, the temperature coefficient (see Section 3.2) is 0.1.

## 4.3 Donwstream Task implementation: Case3

The implementation of this case is similar to the one of Case1 as the objective is to compare both methods.

### 4.3.1 Dataset

Once again, Dataset1 is reused for this task. Moreover, the separation into training, validation, and test sets is the same as for Case1. The preprocessing steps and data augmentation from Case1 are also kept.

### 4.3.2 Hyperparameters

Throughout this case, fine-tuning is performed in two steps. Therefore, it will now be referred to as *two-step fine-tuning*. As explained in Section 3.3, the first step allows training the new head without disrupting the encoder weights. Then for the second step, the entire network is updated. This two-step fine-tuning turned out to be better than directly fine-tuning the whole network with a randomly initialized new head.
Most of the hyperparameters stay the same between the first and the second step of the process. Indeed, for both steps, the batch size is 64, and the number of epochs is 30. About the initial learning rate[1], it increases from the first to the second step, from 0.001 to 0.01.

As a reminder, the hyperparameters have been chosen mainly by trial and error. A common practice about the learning rate while used in a two-step fine-tuning is to decrease its value for the second step not to change much the encoder weights (supposed to already be effective). However, in this work, the opposite process (i.e., increasing the learning rate) has proven to be more effective. The following observations could justify this behavior. The coupling between the backbone and the head is critical [53]. Therefore, using a small learning rate to train the new head for the first time could be favorable to obtain weights compatible with the

[1]The learning rate still follows a step decay scheme during training, hence the term *initial learning rate*.

backbone ones. Then, once the new network becomes one functional entity, training it with a larger learning rate could increase its generalization power [67].

In the end, the choice of hyperparameters stays a complex part of deep network implementation. Although some default values are known for most of them, finding the proper combination for a particular case is not simple and does not always correspond to what is found in the literature.

### 4.3.3 Loss function, Optimizer, and Architectural choices

Concerning all the other aspects of implementation, they are the same as those used in Case1. Indeed, as the aim is to compare both methods, the changes made between them stay minimal.

CHAPTER 5

# Development of a generalizable model

This second task aims to train models to detect pulmonary embolisms (PEs) on any image. This is done through the classification into two categories: PE present or PE absent. After training, these models are considered able to generalize their learning on other datasets.
As said in Chapter 2.3, image classification is a task that has already been studied a lot with supervised methods. However, the purpose of this thesis is to avoid those (see Section 2.3). Therefore, this task has been performed with supervised and contrastive learning to compare both methods. In addition, the networks were challenged to achieve satisfying results while decreasing the amount of labeled data. Similar to Chapter 4, this chapter contains all the implementation details corresponding to both methods.

## 5.1 Supervised implementation: Case2

### 5.1.1 Dataset

In this case, another subset (which will now be referred to as *Dataset2*) of RSNA Pulmonary Embolism CT Dataset is used. Dataset2 contains 19946 images with 15.38% of positive cases used to train the model.
The augmentation of the percentage of positive cases is intended to alleviate the class imbalance problem (see Subsection 4.1.1). By doing so, Dataset2 is a better starting point for this task (further information in Subsection 6.3.1).

The main difference with the first case is the way the dataset is split. Indeed, this

step is essential to avoid any bias in the training process.
For example, if the separation from Case1 was used for this case, the model would be biased as images from the same patient can be in the training, validation, and test set. Therefore, it is imperative to consider the patients when splitting the dataset.
Dataset2 is divided as follows: 80% of the images are for the training set and the remaining 20% for the validation set. However, the separation is also made regarding the patients. That is, images from the validation set are from different patients than those for the training set. Figure 5.1 illustrates this separation. Regarding the test set, other scans from new patients are taken from the RSNA dataset. The test set is composed of 455 scans with 146 positive ones (i.e., 32.08%).

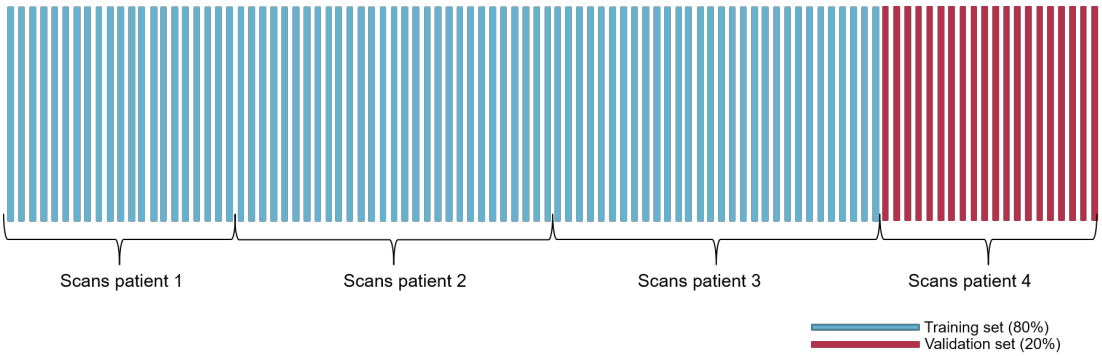



Figure 5.1: Separation of dataset into training and validation sets in Case2.

### Preprocessing

The preprocessing steps (windowing, normalization, and resizing) are the same as those of Case1 (see Subsection 4.1.1). However, one last step has been performed to counter the data imbalance problem. The dataset has been over- and undersampled to balance the class distribution.
Oversampling consists of increasing the frequency of samples from the minority class. Undersampling, on the contrary, aims to decrease the frequency of the samples from the majority class. Therefore, in this case, all the positive images are copied three times, and only two-thirds of the negative images are kept for the training. The resulting training set contains 44.58% of positive cases.

### Data augmentation

Regarding the data augmentation, in this case, it is only used for the training phase, but it does not change from the one used in Case1.

### 5.1.2 Hyperparameters

As for Case1, this implementation uses a step decayed learning rate. However, for some additional experiments presented in Appendix A, the decay is triggered by the iterations and not the epochs. The choice of schedule has a significant influence on the training process. Indeed, in this case, an iteration-based schedule allows reducing overfitting (further information in Appendix A).

Regarding the other hyperparameters, in this case, their values impact less the training process. However, some of theirs do give better results. Therefore, a batch size of 256 has been chosen for a training of 50 epochs.

### 5.1.3 Loss function and optimizer

Concerning the loss function, the binary focal loss (see Equation 4.1) is kept for this case as the dataset is still imbalanced. However, to deal with this problem, another hyperparameter influences the loss function. Indeed, **class weights** have been added to give a higher (resp. lower) weight to the minority (resp. majority) class when computing the loss.
For this implementation, the class weights are inversely proportional to the frequency of the positive and negative classes, according to Equation 5.1:

$$weight_{pos/neg} = \frac{n_{samples}}{n_{classes}\, n_{sample_{pos/neg}}} . \tag{5.1}$$

In the above, $n_{samples}$ is the number of images in the training set, $n_{classes}$ is the number of classes and $n_{samples_{pos/neg}}$ is the number of positive (resp. negative) images in the training set.

The optimizer used for this case is the **Adaptive moment estimation** (Adam) optimizer [68]. The main benefit of Adam (over SGD, for example) is that it computes adaptive learning rates throughout the training, making itself an efficient algorithm. Indeed, Adam is «robust and well-suited to a wide range of of non-convex optimization problems »[68], which makes it a commonly used optimizer.

### 5.1.4 Architectural choices

For this case, the architecture is the same as the one described in Subsection 4.1.4. However, a modification has been made on the last layer, once again, to overcome the problem of dataset imbalance. A bias is set to initialize the final layer weights. This way, the loss value of the first iterations is not destabilized, possibly speeding

up the convergence.
The bias has been chosen according to [61], and its value is computed following Equation 5.2.

$$\text{bias} = -\log(\frac{1-\pi}{\pi}) \quad \text{with } \pi = 0.01. \tag{5.2}$$

## 5.2 Implementation Case4

The contrastive implementation of Case4 is identical to the one described in Section 4.2, except that the pre-training is performed on Dataset2.

Regarding the downstream task, once again, the implementation for the fine-tuning process intends to be close to the one of the supervised learning. In this case, there are no changes compared to what is described in the first section of this chapter. Indeed, compared to the fine-tuning from Case3, this one is performed in one step, for which the hyperparameters correspond to those of the supervised training.

CHAPTER 6

# Results

In this chapter, the results from Chapters 4 and 5 are presented and discussed. The metrics used to evaluate models performances are defined in the first section. Then, the second one is devoted to the results of the first task (i.e., annotation enrichment). It starts with the supervised results, then the contrastive ones, and ends with the comparison of the two. Eventually, the third section focuses on the results of the second task (i.e., general classification). This section is divided into two parts: the supervised results and the methods comparison.

## 6.1 Metrics presentation

The choice of the metrics is critical as the interpretation of the model performance depends on it. A wrong choice could lead to a poor evaluation of the network. Therefore, comparing several metrics is an efficient way to evaluate a network in an unbiased manner.

### 6.1.1 Confusion matrix

The confusion matrix is a great starting tool to determine the performance of a model. This metric speaks for itself when it comes to interpreting it. Indeed, the matrix directly displays the number of true and false predictions in the form of a four-entry table (see Figure 6.1).

The true positive (resp. negative) cases are the correctly predicted positive (resp. negative) images. In opposition, the false positive (resp. negative) cases are the images classified as positive (resp. negative) but truly negative (resp. positive).

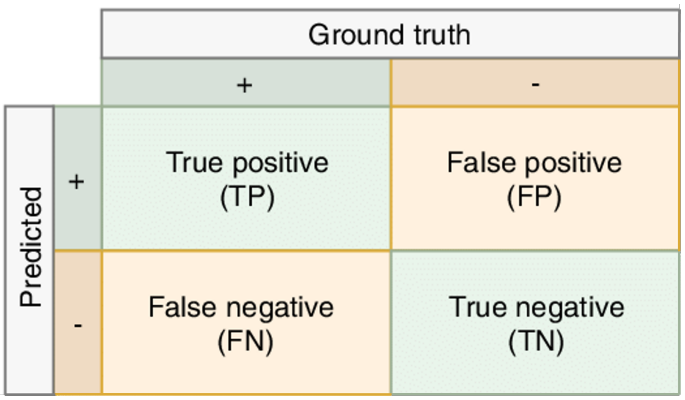



Figure 6.1: Confusion matrix [69].

In addition to being a relevant source of information, the confusion matrix is also the basis from which the following metrics are defined.

### 6.1.2 Accuracy

The accuracy is a common and easy to interpret evaluation metric. It calculates the percentage of correct predictions according to Equation 6.1.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}\,. \tag{6.1}$$

However, its drawback is that it can be misleading if not well interpreted. For example, in this case, as said in Subsection 4.1.1, the dataset is strongly imbalanced. Therefore, the model could always predict no pulmonary embolism no matter the input and still have more than a 95% accuracy which, at first sight, seems convincing.
This example shows the importance of comparing multiple metrics, hence avoiding a wrong straightforward evaluation.

### 6.1.3 F1-score

The F1 score (Equation 6.2) is the harmonic mean of the precision (how many of the positive predictions are correct) and recall (how many of the positive cases are classified correctly).

$$F1score = 2 \cdot \frac{precision \cdot recall}{precision + recall}\,, \tag{6.2}$$

with

$$precision = \frac{TP}{TP + FP}\,, \tag{6.3}$$

$$recall = \frac{TP}{TP + FN}\,. \tag{6.4}$$

The advantage of the F1-score is that it deals with the imbalanced problem by emphasizing the correct predictions of the under-represented class.

### 6.1.4 ROC/AUC

ROC stands for Receiver Operator Characteristic. This evaluation metric takes the form of a graph that displays the True Positive Rate (TPR) against the False Positive Rate (FPR) for several thresholds.
The TPR (also called sensitivity) is the proportion of correctly predicted positive samples. In opposition, the FPR is the fraction of incorrectly classified negative samples. TPR and FPR are defined according to the following formulas:

$$TPR = \frac{TP}{TP + FN}\,, \tag{6.5}$$

$$FPR = \frac{FP}{FP + TN}\,. \tag{6.6}$$

The threshold is the value above (resp. under) which a prediction is considered positive (resp. negative).
Figure 6.2 displays an example of a ROC curve. Its interpretation is straightforward. A model is judged efficient when the TPR is maximized while the FPR is minimized. The perfect point on the curve is the upper left corner, for which all the predictions are right. Therefore, any curve that tends to this point is a sign of a good model. This evaluation is quantified by the Area Under the Curve (AUC). The larger the AUC, the better.
**NB**: A network with an AUC of 0.5 (i.e., ROC on the «random guess» line) means that its predictions are made randomly (like the flipping of a coin, for example).

This metric is an efficient approach to compare models. In addition, it also provides the threshold that optimizes the network performance. Indeed, the best point on the curve is the one for which the difference between the TPR and FPR is minimal. Therefore, the threshold corresponding to this point is the most suited for the network in question.

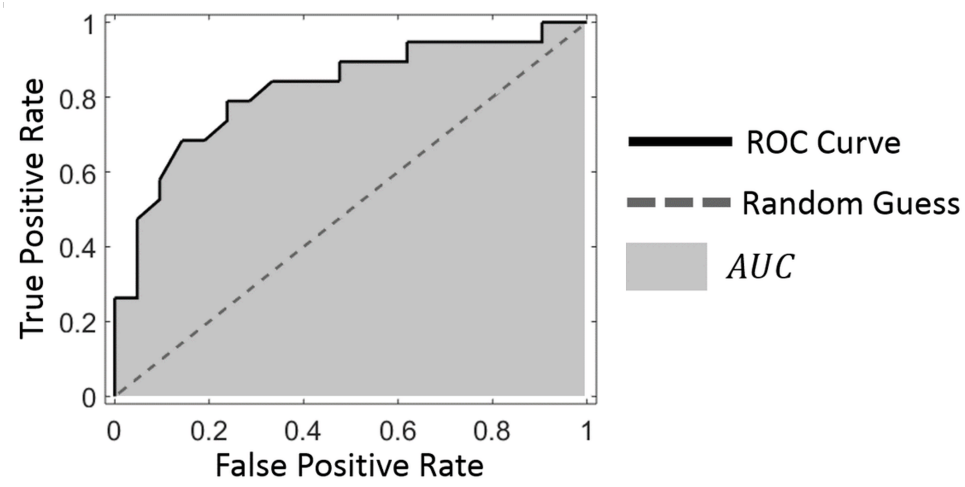



Figure 6.2: Example of ROC curve [70].

## 6.2 Annotation enrichment - Cases 1 & 3

### 6.2.1 Preliminary remarks

#### Randomness of simulations

Even though the chosen hyperparameters are fixed, there are some other aspects of the training process that bring certain randomness, such as the model weights initialization, the dropout layer, the data augmentation, etc. In addition, in this case, the dataset separation is also one of these aspects.

Theoretically, even though challenging, it is possible to control those aspects to obtain reproducible results. However, it is usually customary to do several tests with the same hyperparameter combination to deal with those random aspects. Then, either the mean of all tests can be presented, or the best model (not only in terms of hyperparameters). For example, **K-fold cross validation** [71] is a procedure used against the randomness of the dataset separation. The idea behind this technique is to split the dataset into multiple parts, then to use each part once as validation set while the others form the training set.

Nevertheless, in practice, fully reproducible tests are hard to implement, and for the sake of time, all the tests have been performed only once. Therefore, it is essential to keep in mind throughout this section that even if the results give a first sight of the studied behavior, it may be possible that the best results of one test are compared to the worse ones of another.

#### Denomination of tests

In the following subsection, different experiments are presented. For clarity, each of the latter is assigned a name listed in Table 6.1.

| Sampling rate | $\frac{9}{10}$ | $\frac{4}{5}$ | $\frac{3}{4}$ | $\frac{2}{3}$ | $\frac{1}{2}$ | $\frac{1}{3}$ | $\frac{1}{4}$ |
|---|---|---|---|---|---|---|---|
| Denomination | Test90 | Test80 | Test75 | Test66 | Test50 | Test33 | Test25 |

Table 6.1: Denomination of the tests of Cases 1 and 3.

The different tests are listed according to their sampling rate (i.e., the rate at which images exploited for training are sampled from Dataset1). As explained in Subsection 4.1.1, a sampling rate of 9/10 means that nine images out of ten are kept for the training. The dataset is thus split into two parts: 90% for the training stage and 10% for the test set. When it comes to a sampling rate of 1/3, one image out of three is sampled, which gives a set of 33,33% for the training process.

**NB**: The test set also changes according to the sampling rate for the above seven experiments. Therefore, although they are compared throughout this chapter, they should be considered as separated entities as the comparison between them is less relevant with different test sets. However, comparisons with contrastive learning are, as both methods are applied to perform the same tests, thereby evaluated on the same test sets.

### 6.2.2 Supervised results - Case1

#### Metrics interpretation

Both graphs in Figure 6.3 show the training (blue curve) and validation (orange curve) of Test90 for two different metrics, that is, F1-score and binary accuracy.

The result the most straightforward to interpret in the figure is the accuracy. In this case, this value can already give a piece of good information. Indeed, the accuracy is high even when taking into account the imbalanced aspect of the dataset. Its value of 0.9961 proves that the model can classify, without mistakes, both negative and positive samples (otherwise, the accuracy would have reached a maximum of 0.96, corresponding to the fraction of negative images).

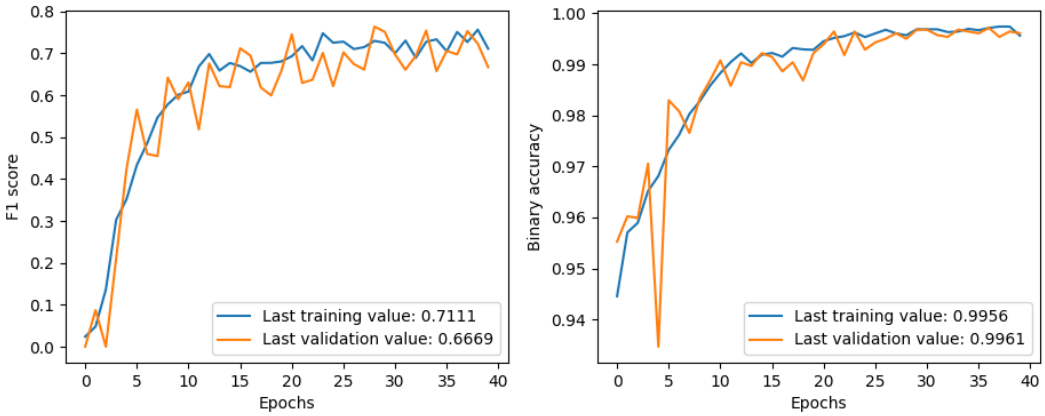



Figure 6.3: Metric values for Test90 of Case1.

The other metric used when training the model is the F1-score. Its interpretation is more complicated. Indeed, theoretically, the value of F1-score is between 0 and 1, 1 being the maximum, meaning no error in the predictions. However, a higher value does not always mean better results (further explanations in Appendix B). According to its definition (Equations 6.2, 6.3, and 6.4), F1-score is mainly impacted by the positive samples. Therefore, it is necessary to consider the distribution of the latter throughout the training process to interpret the metric.

*The following results concerning the F1-score have been computed afterward to illustrate the explanation. They come from another training with similar hyperparameters. Therefore they are not directly linked with the upper results, but they still give an insight into the F1-score interpretation.*

In this case, as Dataset1 is highly imbalanced (4.21% of positive images) and the batch size is 32, there is quite a high percentage of batches that do not contain any positive image. For those, the F1-score is automatically 0 hence impacting the overall metric computed as a mean of each F1-score per batch. Therefore, knowing the distribution of the positive images during the training process provides the maximum bound for the F1-score. With this bound, the results are compared to a more realistic value than the theoretical maximum. The following example illustrates the higher bound calculation. Figure 6.4 shows the distribution of positive images for one epoch.

Knowing this distribution, the maximum F1-score for this epoch can be computed, considering that for the 100 batches without positive images, the F1-score is 0, and for the 250 others, the score is 1. The maximum value is thus 0.7143, which is way below one and closer to the validation one obtained with the model. This operation

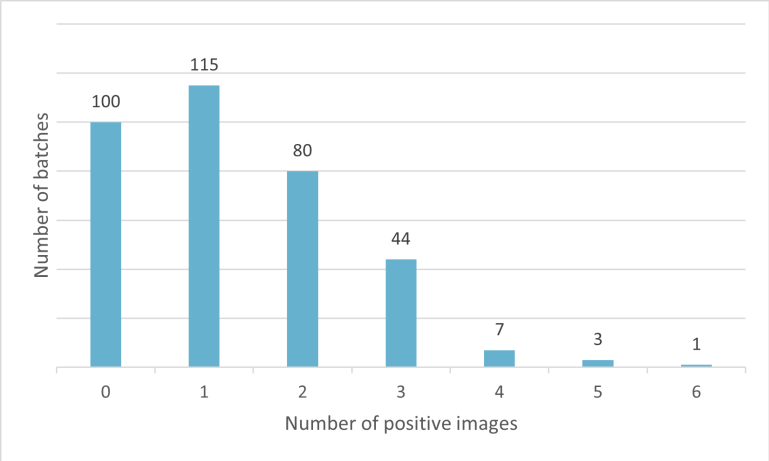



Figure 6.4: Distribution of the positive images of the training set across all batches of one epoch.

has been repeated for all epochs. Therefore, the overall maximum F1-score for the entire training (mean of the maximum value per epoch) is 0.7364.

From then on, even though this does not give much concrete information about the model yet (except that the gap of the F1-score is not as big as intuitively thought), it can explain how low F1-scores can already be good and sometimes even better than higher ones (see Appendix B).

**NB**: Another way of computing the F1-score allows avoiding these null scores. Instead of updating the F1-score at each batch, a better way would be to update the confusion matrix. The F1-score would therefore be computed at the end of each epoch.

Anyway, metrics are not that easy to interpret and can even be misleading. Therefore, using several ways of displaying the results can help. The confusion matrix is a good one as it directly shows the number of correct and wrong predictions on the test set. For the training corresponding to Figure 6.3, the confusion matrix is shown in Figure 6.5.

The confusion matrix gives a more direct way of interpretation. Indeed, when looking at it, the model performance can be highlighted. In this case, the model seems quite efficient as it is only wrong for seven images out of 1556. These results correspond to an accuracy of 0.9955 and an F1-score of 0.9489 (to be compared to a maximum of 1 as the test set is processed as one entity).

| | |
|---|---|
| **TN**<br>1484 (99.8%) | **FP**<br>3 (0.2%) |
| **FN**<br>4 (5.8%) | **TP**<br>65 (94.2%) |

Figure 6.5: Confusion matrix for Test90 of Case1.

Finally, to highlight the impact of the training set reduction, Figure 6.6 shows the F1-score according to the sampling rate. As expected, the metric value decreases with the training set. This graph also brings out a possible outlier due to the randomness of the simulations since the F1-score of Test75 is particularly low compared to the curve tendency.
These results do not directly mean that the last sampling rate is the worse. It depends on the below uncertainty evaluation. Indeed, the F1-score drop with small training sets can be compensated by the gain in the annotation work.

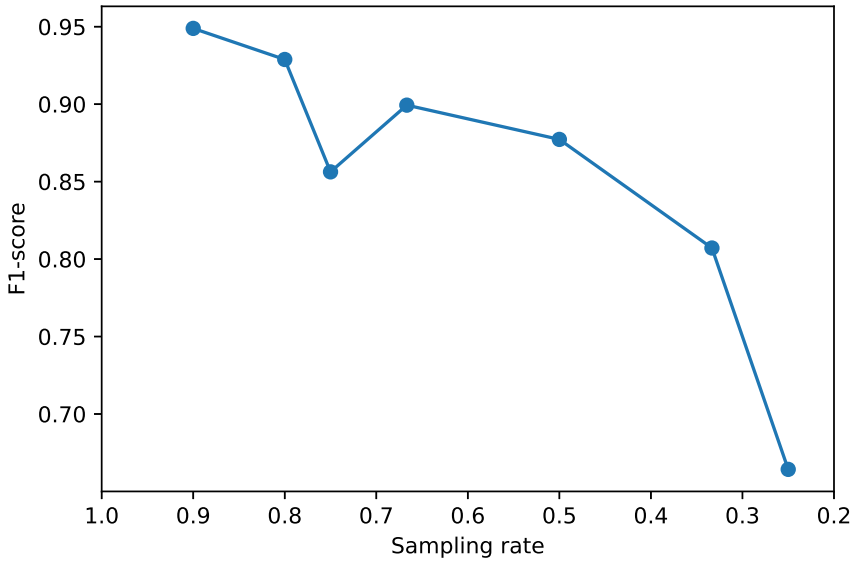



Figure 6.6: F1-score according to the sampling rate for Case1.

### Uncertainty evaluation

All the above results can be confusing and abstract. This is why some methods are used to highlight the model uncertainty and to help to present the results

more understandably. In this case, as explained in Subsection 4.1.1, test-time augmentation has been performed, allowing to classify images of the test set into three categories, namely, **True Predictions**, **False Predictions** and **Uncertain Predictions**. This classification depends on the chosen thresholds, as visible in Figure 6.7. As a reminder, the upper threshold defines the value above which a prediction is considered positive and the lower one, the value under which a prediction is negative. The space between both thresholds contains the uncertain predictions.

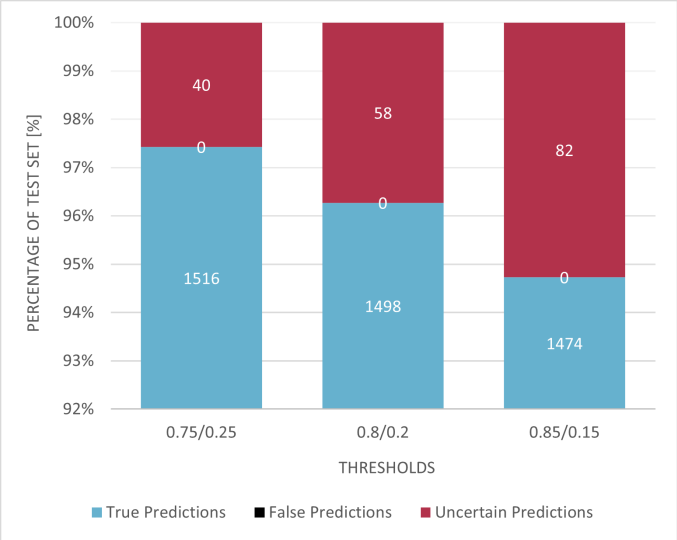



Figure 6.7: Test-time augmentation results according several thresholds for Test90 of Case1.

As expected, the higher (resp. lower) the upper (resp. lower) threshold, the more uncertain the predictions. These thresholds are decisive when it comes to finding the best possible model. Indeed, in this case (Figure 6.7), there is no need to choose highly restrictive thresholds as the network does not make mistakes even with the less restrictive ones (above 0.75 and under 0.25). However, when sampling more and more images from the training set, those thresholds need to be adapted to keep a model without any wrong predictions. Figure 6.8 shows the different thresholds applied for two more examples, respectively, Test75 (sampling rate of 3/4) and Test25 (sampling rate of 1/4).

Figure 6.8 highlights the need for adaptive thresholds to maintain a model that is always right in its predictions. While for Test90 (sampling rate of 9/10), thresholds of 0.75 and 0.25 are enough, when increasing the sampling rate, more restrictive

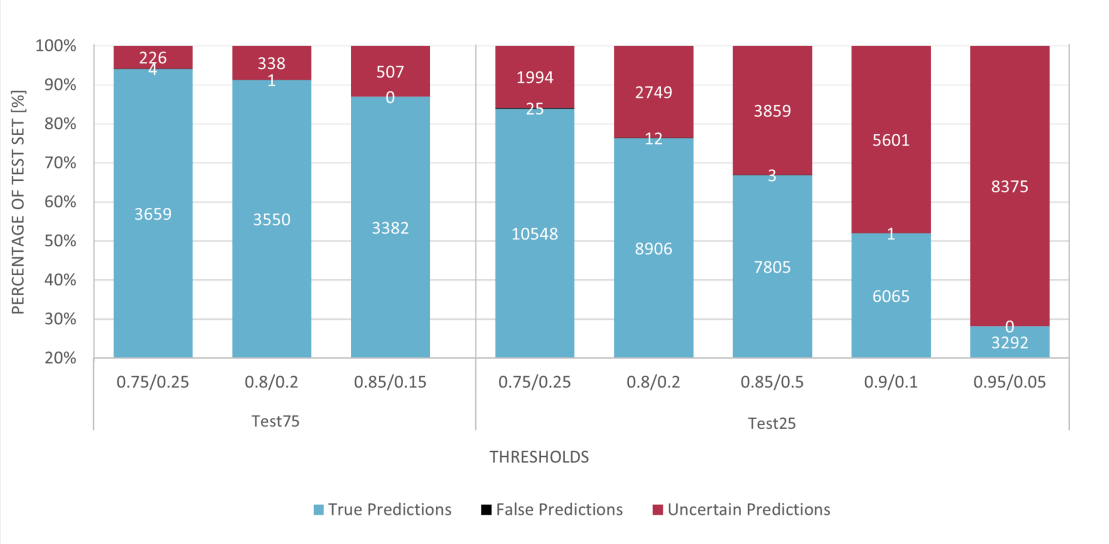



Figure 6.8: Test-time augmentation results for Test75 and Test25 of Case1.

thresholds are needed: the pair 0.85/0.15 for Test75 and 0.95/0.05 for Test25.

The choice of both thresholds also depends on the required accuracy of the model. If there is a margin of error in the desired accuracy, those thresholds can be wider. This allows decreasing the number of uncertain predictions (which have to be rechecked by the medical experts) while admitting that a small percentage of images will be permanently wrongly annotated. Such a choice depends on the context in which the model is deployed. However, for the sake of simplicity, the following comparisons will be made for models reaching a perfect accuracy.

**NB**: In this case, the following tests demonstrate more the possible results that can be obtained than real-life results. Indeed, in reality, the test set is not annotated, which means that the thresholds chosen to reach a perfect accuracy cannot be known.

Although, Figures 6.7 and 6.8 are convenient to illustrate the choice of a relevant threshold, they can be misleading when comparing one test to another. Indeed, even though the percentage of uncertain predictions is high when increasing the sampling rate, on the other hand, the number of images in the training set (those initially annotated) decreases. This is illustrated in Table 6.2, which contains the results of Case1 for all the tests. As said previously, the comparison is made for a threshold that enables the model to reach a perfect accuracy.

| | Test90 | Test80 | Test75 | Test66 | Test50 | **Test33** | Test25 |
|---|---|---|---|---|---|---|---|
| Percentage of images initially annotated | 90 | 80 | 75 | 66.66 | 50 | **33.34** | 25 |
| Percentage of uncertain predictions | 0.25 | 1.52 | 3.26 | 11.11 | 18.91 | **19.61** | 53.84 |
| Total percentage of images manually annotated | 90.25 | 81.52 | 78.26 | 77.77 | 68.91 | **52.95** | 78.84 |
| Thresholds | 0.75/0.25 | 0.85/0.15 | 0.85/0.15 | 0.95/0.05 | 0.95/0.05 | **0.9/0.1** | 0.95/0.05 |

Table 6.2: Summary table of the results of Case1. The bold column corresponds to the best test.

Results of Table 6.2 show that **it is possible to reduce by almost half the work of annotation**. Indeed, for Test33 (sampling rate of 1/3), the training set only contains a third of Dataset1, considerably reducing the annotation work. Then, with that training set, it is possible to train a model such that it can annotate the remaining images with about 80% of certainty. Altogether, only 52.95% of Dataset1 have to be manually annotated (that is, the training and rechecked images).

### 6.2.3 Contrastive results - Case3

The impact of representation learning is mostly visible while performing the downstream task. However, it is still possible to extract some results illustrating the smooth running of the contrastive pre-training. This can be done through the contrastive loss curve and a visualization process.

#### Contrastive loss

Knowing the objective of the training process (i.e., minimizing a loss function), a look at the loss curve can be the first indicator of smooth running. Figure 6.9 displays this curve, obtained with the training process detailed in Section 4.2, for Case3 with 75 epochs.
Although it does not provide much information about the performance on the downstream task, this curve can show at least one impact of hyperparameter on the training. Indeed, in Figure 6.10, the benefit of a long training is easy to see as the loss continues to decrease with the epochs.

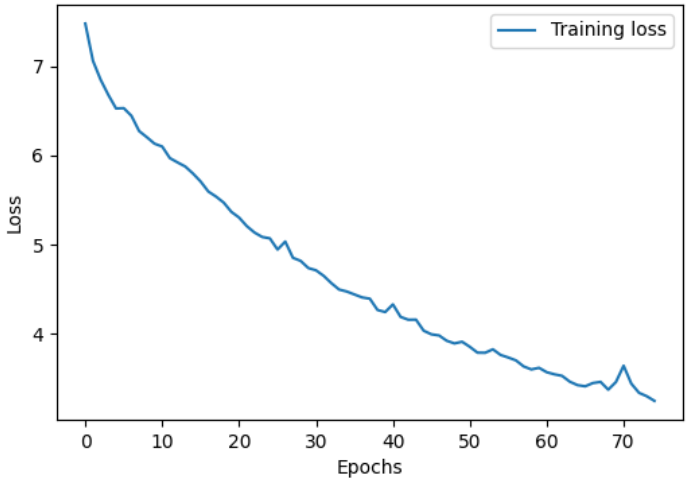



Figure 6.9: Contrastive loss for Case3 with 75 epochs.

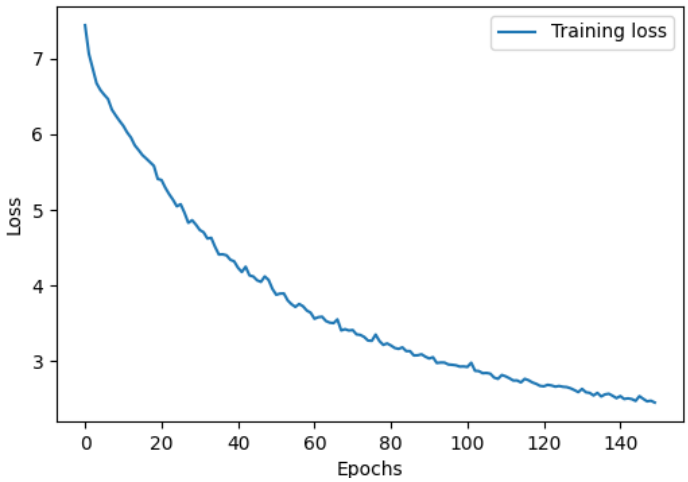
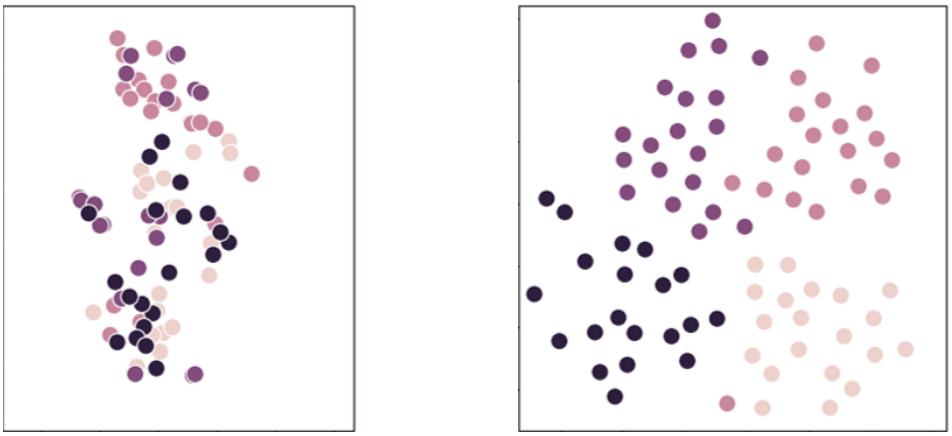



Figure 6.10: Contrastive loss for Case3 with 150 epochs.

### Representation vectors visualization

The other way to analyze contrastive learning results is to visualize the representation vectors in a 2D space. Indeed, after training, representations of positive pairs should be close to each other while being distant from the negative ones.
The visualization process is a problem of dimensionality reduction from the representation space to a 2D space. This mapping aims to maintain the same distance ratio between images, and it is performed with the **t-distributed stochastic neighbor embedding (t-SNE)** algorithm [72].
This algorithm has been applied to four images, one from the training set and three new ones. To see the benefit of the training process, instead of being augmented

twice, each image (represented by the four colors in Figure 6.11) has been augmented twenty times. The clusters formed by the contrastive training are therefore easier to see. Figure 6.11 shows the representation vectors, taken from the last layer of the encoder, before and after contrastive learning for the same input images.



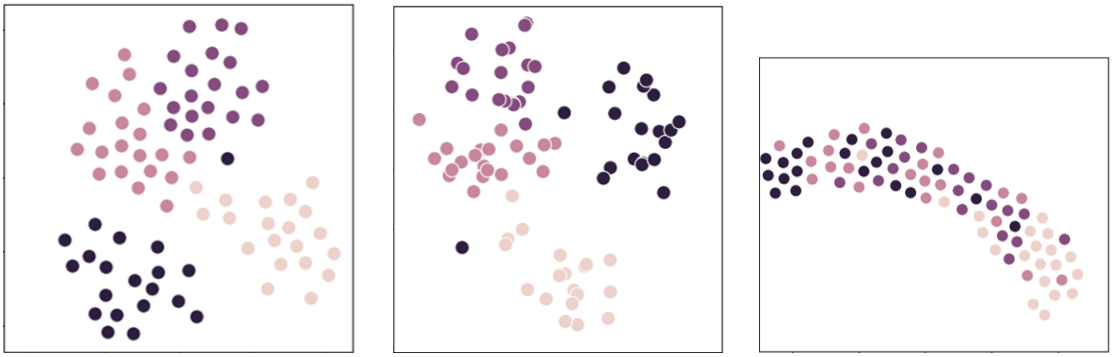
(a) t-SNE before contrastive learning. (b) t-SNE after contrastive learning.

Figure 6.11: Visualization of the images representations for Case3 (75 epochs). Each point from the same color representing a different view of the same image.

Once again, those representations do not predict which results will be obtained for the downstream tasks. However, as for the contrastive loss, the visualization can give some good indications about the smooth running of the process. Indeed, while the images are initially randomly placed in the representation space (Figure 6.11a), after the contrastive pre-training, the clusters of augmented views belonging to the same image are well distinct, as visible in Figure 6.11b.

The visualization can also provide information about the best layer to keep as the last layer when transferring or reusing the model (see Section 3.3). Indeed, the representation is different according to the layer from which it comes. Hence this visualization can help to choose the best layer. Figure 6.12 shows the visualization of the same images after contrastive learning (for Case3 with 150 epochs) but taken from the last layer of the encoder and the first and the second layer of the non-linear head.

Although, it is quite hard to evaluate the difference between the end of the encoder (Figure 6.12a) and the first layer of the head (Figure 6.12b), the representation at the end of the head is distinctly worse. This proves that the head (or at least its last layer) should be discarded before reusing the network for any downstream



(a) Last encoder layer. (b) First head layer. (c) Second head layer.

Figure 6.12: Visualization of the image representations for Case3 (150 epochs) from different layers of the contrastive network.

task. Concerning the choice of keeping the first layer of the head or not, it has been studied throughout the second SimCLR paper [73]. The authors of this one stand that it would lead to better results to use a wider head (3 layers and more) and keep the first layer of it.

Finally, although it is not an accurate efficiency metric, t-SNE can still provide a convenient first approach to estimate if contrastive learning has been well performed or not.

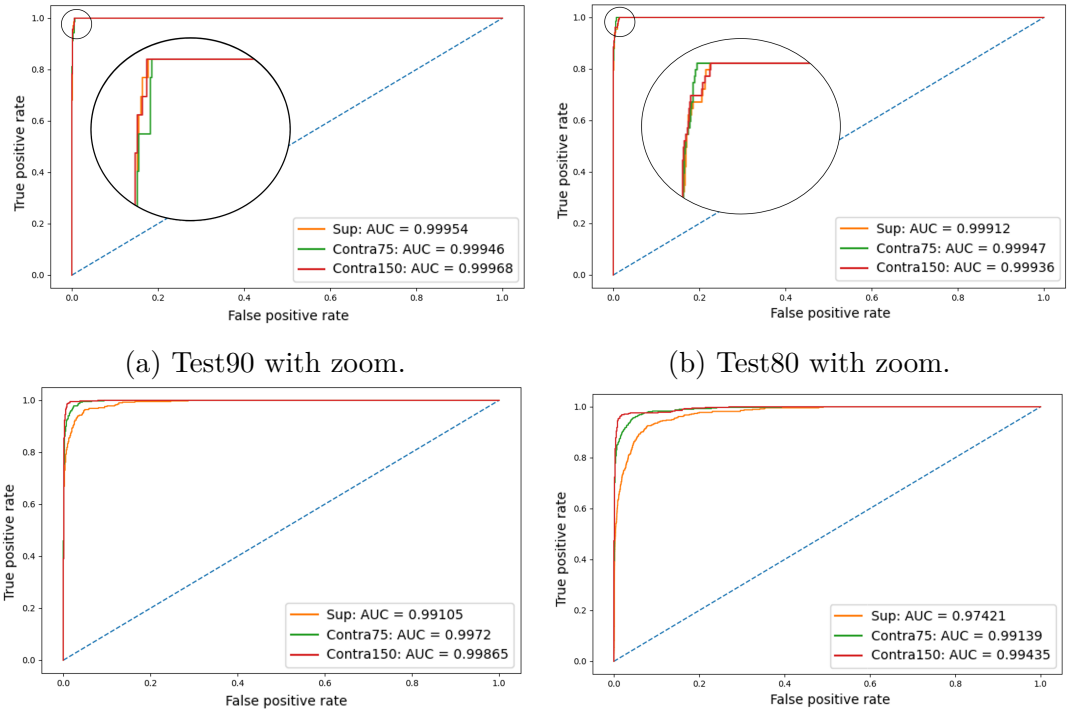
### 6.2.4 Methods comparison

As this approach (i.e., annotation enrichment) already gives good results with supervised learning, its implementation via contrastive learning is mainly a way to see if this method can improve the results of different downstream tasks. Therefore, the results of the fine-tuning post contrastive learning (Case3) are not presented alone as the supervised ones but directly compared to the latter.

#### Metrics interpretation

The ROC/AUC metric has first been chosen to compare both methods because of its visual simplicity. In addition, as contrastive learning is supposed to benefit from longer training, a third set of experiments has been run with a contrastive pre-training of 150 epochs. Therefore, two comparisons are made. The first one is between supervised and contrastive learning, while the second is between two implementations of this last method.

Figure 6.13 depicts the three ROC curves for several tests.




(a) Test90 with zoom.

(b) Test80 with zoom.

(c) Test33.

(d) Test25.

Figure 6.13: ROC/AUC curves comparing the implemented method for several tests.

These ROC curves show that contrastive learning constantly outperforms supervised learning. Indeed, for those tests, which are the extreme ones of the series, the results with either one of the contrastive pre-trainings predominate. In addition, except for Test80, the pre-training of 150 epochs gives better results than the shorter one.

Another comparison, depicted in Figure 6.14, has been performed with the F1-score metric for all the sampling rates.
Firstly, this graph highlights the fact that the benefit of contrastive learning increases with the reduction of the training set, which is the desired property of the contrastive pre-training.
Secondly, in agreement with the ROC curves, except for Test80, a longer pre-training always results in an F1-score improvement.
Finally, just as the supervised Test75, Test80 from the contrastive pre-training of 150 epochs is out of the curve tendency. This behavior suggests that Test80 could be impacted by the randomness of the simulations and therefore improved.

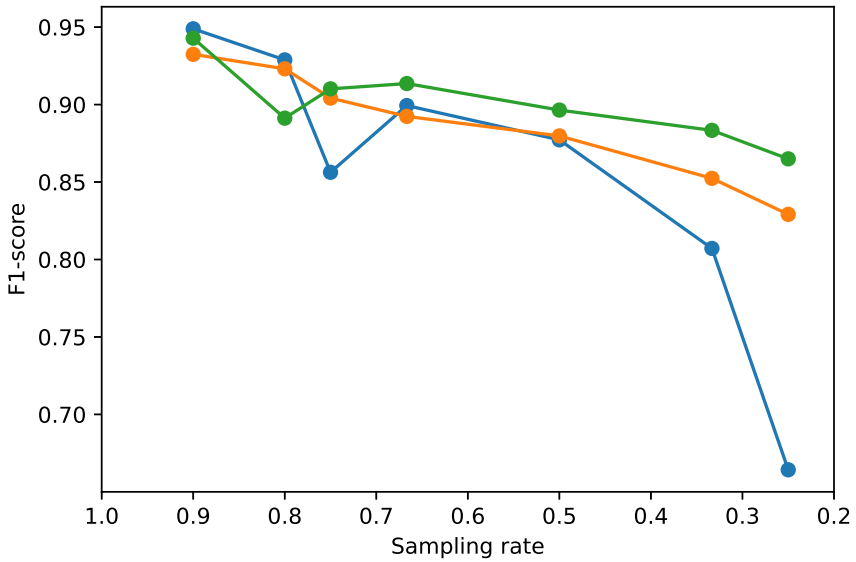



Figure 6.14: F1-score according to the sampling rate for Cases 1 and 3.

### Uncertainty evaluation

The final comparison is visible in Figure 6.15. This one shows the total percentage of manually annotated images (i.e., those used for the training and the rechecked ones) for both methods and different tests/sampling rates. As before, the comparison is made for thresholds that give all the models a perfect accuracy considering the test sets labels (hence the color code). Therefore, the tests are more representative of the results that can be theoretically achieved, since in a real-life project, the labels of the test sets are unknown.

| Method | Test90 | Test80 | Test75 | Test66 | Test50 | Test33 | Test25 |
|---|---|---|---|---|---|---|---|
| Supervised learning | **90.25** | 81.52 | 78.26 | 77.77 | 68.91 | 52.95 | 78.84 |
| Contrastive learning (75 epochs) | 90.37 | 81.06 | **77.04** | **68.55** | 57.19 | **49.85** | 57.01 |
| Contrastive learning (150 epochs) | 90.27 | **80.83** | 77.79 | 68.91 | **54.14** | 55.51 | **49.34** |

Thresholds
0.75/0.25
0.8/0.2
0.85/0.15
0.9/0.1
0.95/0.05

Figure 6.15: Percentage total of images that have to be manually annotated according to the method and the sampling rate. The bold values are the best of each test. The color code refers to the required thresholds to reach a perfect accuracy.

The first conclusion drawn from these results is that, except for Test90, contrastive learning outperforms supervised learning. The advantage lies either in the percentage of annotated images when the same threshold is used or in the fact that the models can achieve a perfect accuracy with wider thresholds. The benefit of contrastive learning is more visible as the sampling rate increases (except for Test33).

As visible in Figure 6.15, the longest contrastive training (150 epochs) constantly outperforms the 75 epochs one, provided that the thresholds are the same (for Tests 90, 80, 50, and 25). Indeed, a more restrictive threshold always leads to more uncertain images and thus harder comparison in terms of value (i.e., percentage of annotated images).

Eventually, considering the aim of this approach, contrastive learning allows reducing, even more, the work of annotation and the training costs as the best performance corresponds to Test25.

Another advantage of contrastive learning is visible in Figure 6.16. The last one displays the percentage of positive images that are rightly predicted among all the positive images. As Dataset1 is highly imbalanced, the ability to correctly predict positive and negative images reflects a training process that is more robust against the imbalance problem.

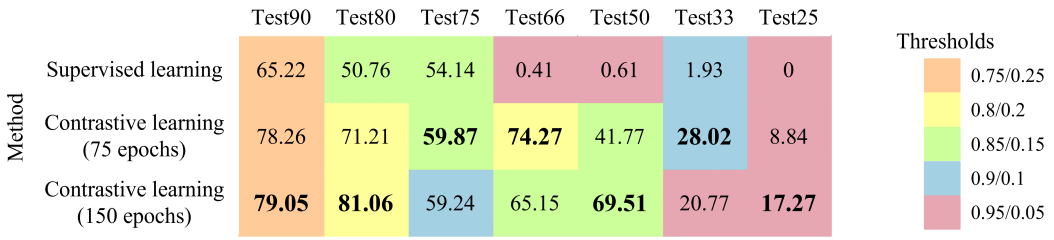



Figure 6.16: Percentage of positive images correctly predicted among all the positive images of each test set according the methods and the sampling rate. The bold values are the best of each test. The color code refers to the required thresholds to reach a perfect accuracy.

This time, contrastive learning outperforms supervised learning for all tests. This could mean that contrastive learning is more robust against the imbalance problem. However, it is important to keep in mind that the batch size used for the fine-tuning post contrastive learning is twice the size of the supervised one. This means that there are fewer batches without positive images. Therefore, the batches for the fine-tuning are a better representation of the entire dataset.

Once again, the batch size has been chosen to obtain the best possible results in both cases. This choice could therefore be based on simulations that could have been outliers due to the randomness.

**Areas for improvement**

As said in Subsection 6.2.1, the above comparisons might sometimes be biased due to the random aspects of the training process. Therefore, it would be interesting to perform more tests for several reasons.

1. To see if supervised learning can reach a perfect accuracy at the same thresholds as contrastive learning and compare the values in this case.
2. To see if the contrastive training of 150 epochs can outperform the shorter one for all sampling rates provided it can reach the same thresholds.
3. To see if the best result can still be improved (if Test33 of contrastive learning 150 outperforms the result of the shorter one, for example).
4. To see if supervised learning (resp. contrastive learning) can achieve as good results with a batch size of 64 (resp. 32) to be able to compare the percentage of correctly predicted positive images with the same batch size.

## 6.3 General classification - Cases 2 & 4

This second task consists of training a network so that it can perform the classification of CTPA scans containing or not pulmonary embolisms in a general manner (i.e., for any images not used during the training). This kind of task is representative of a typical deep learning purpose, which is to develop a network able to generalize its learning the best to be further deployed for daily life problems. For the presentation of this task, there will be a greater focus on the approach and the path taken to get to the current state.

### 6.3.1 Supervised results - Case2

Table 6.3 summarizes the main experiments that have led to the choice of hyperparameters presented in Chapter 5. The first column of the table contains the numbers that will be used to reference the different experiments. The central part focuses on the hyperparameters: their usage, their value, etc. Finally, the columns on the right contain the values of different metrics computed from the test set. The F1-score and accuracy have been computed for the thresholds maximizing the AUC.

| | Dataset | Initial LR | Batch size | Optimizer | Data augmentation | Dropout | Sampling | Class weights | Bias | AUC | F1-score | Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4.21% (15556 img) | 0.01 | 256 | SGD | ✓ | ✗ | ✗ | ✗ | ✗ | 0.67589 | 0.55422 | 0.59341 |
| 2 | **16.09% (15555 img)** | 0.01 | 256 | SGD | ✓ | ✗ | ✗ | ✗ | ✗ | 0.7544 | 0.59638 | 0.70549 |
| 3 | 16.09% (15555 img) | 0.01 | 256 | SGD | ✓ | ✓ | ✗ | ✗ | ✗ | 0.79494 | 0.67838 | 0.71868 |
| 4 | 16.09% (15555 img) | 0.01 | 256 | SGD | ✗ | ✓ | ✗ | ✗ | ✗ | 0.82152 | 0.67156 | 0.70549 |
| 5 | 16.09% (15555 img) | **0.1** | 256 | SGD | ✓ | ✓ | ✗ | ✗ | ✗ | 0.81272 | 0.67991 | 0.71648 |
| 6 | 16.09% (15555 img) | 0.01 | **64** | SGD | ✓ | ✓ | ✗ | ✗ | ✗ | 0.82963 | 0.69468 | 0.76044 |
| 7 | 16.09% (15555 img) | 0.01 | 64 | SGD | ✓ | ✓ | ✗ | ✓ | ✓ | 0.82143 | 0.72386 | 0.77363 |
| 8 | 16.09% (15555 img) | 0.01 | 64 | **Adam** | ✓ | ✓ | ✗ | ✓ | ✓ | 0.83491 | 0.66194 | 0.68571 |
| 9 | 16.09% (15555 img) | 0.01 | 64 | Adam | ✓ | ✓ | ✓ | ✓ | ✓ | 0,89859 | 0,75964 | 0,82198 |
| 10 | 16.09% (15555 img) | 0.01 | **256** | Adam | ✓ | ✓ | ✓ | ✓ | ✓ | 0,89795 | 0,80644 | 0,86813 |
| 11 | **15.38% (19946 img)** | 0.01 | 256 | Adam | ✓ | ✓ | ✓ | ✓ | ✓ | 0.92676 | 0.81286 | 0.85934 |

Table 6.3: Table of Case2 experiments. The gray line corresponds the best one. The bold values mark the changes between experiments.

For the sake of clarity, few details have to be specified before analyzing this table. Concerning the table entries, regarding the dataset, the percentage information refers to the fraction of positive images. Then, the learning rate of these tests follows a step decay function based on the epochs. Finally, a momentum of 0.9 is used with the SGD optimizer. The other hyperparameters values correspond to their description in Chapter 5.
About the experiment analysis, at the time the tests were conducted, the comparison between them was mainly based on the training and validation metrics (F1-score and accuracy), the shape of their curves, and the confusion matrix computed on the test set (for a threshold of 0.5). The metrics present in the table have been computed more recently to verify the relevance of the choices.
Finally, concerning the approach, each experiment during the tuning of hyperparameters has only been performed once. A bias due to the randomness of the training process is thus present.

According to Table 6.3, the path to the choice of hyperparameters is the following:

$1 \rightarrow 2$: First, attempts with a dataset containing the same distribution of positive images as the complete one have been made. However, the results (especially the graphs) were not convincing. Therefore, another dataset with more positive samples has been used as it represents a better starting point. Graphs of experiments one and two can be found in Appendix C.1.

$2 \rightarrow 3$: As the training stage was suffering from overfitting, a dropout layer has been added to the head of the network (see Subsection 5.1.4). The benefit is visible

for all the metrics.

3 → 4: From step three, multiple experiments have been conducted. Regarding the data augmentation, although the AUC increases when not used, the training process is limited. Indeed, as shown in Figure 6.17, the network learns the training set quickly, causing the plateau on the graphs. Therefore, no significant learning is made.

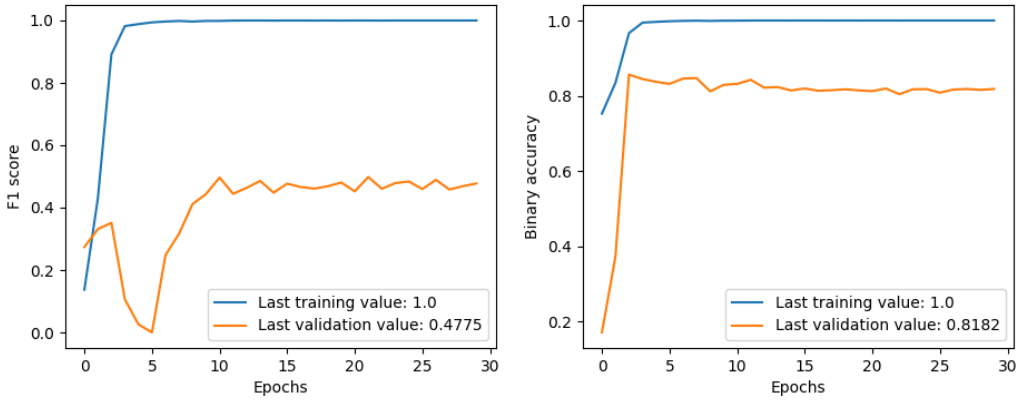


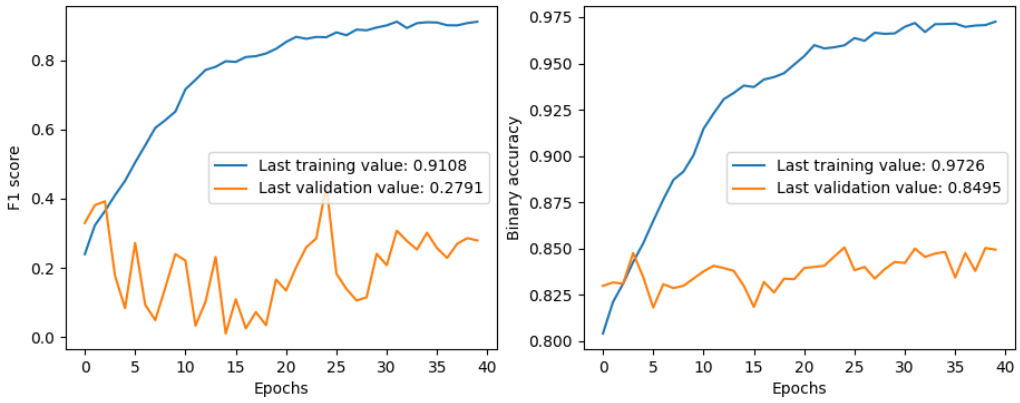

Figure 6.17: Learning curves of experiment 3 of Table 6.3.

3 → 5: The choice between the two initial learning rates is less justified when considering the metrics on the test set. Indeed, they give almost similar F1 score and accuracy values, while the AUC is significantly better for 0.1. However, considering the learning curves and other conducted experiments, on average, 0.01 gave better results (see Appendix C.2), hence this choice.

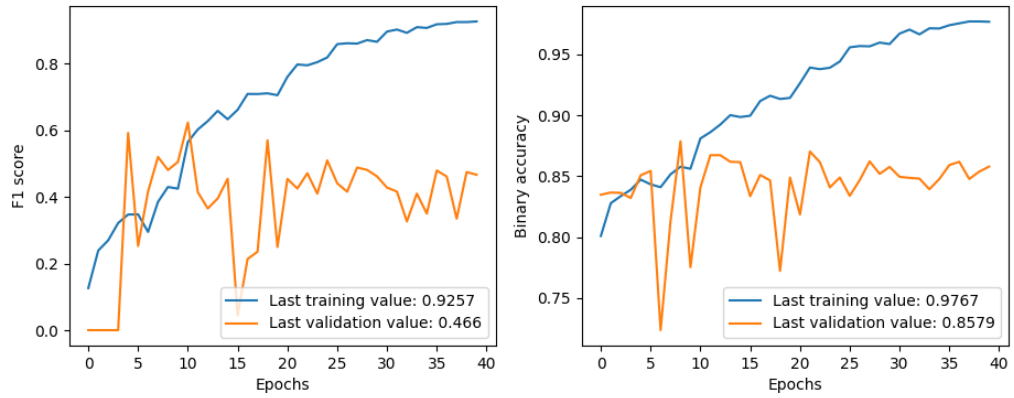
3 → 6: This time the modified hyperparameter is the batch size. Metrics values show that at this point, batches of 64 images are giving better results. Hence the following tests with this batch size.

6 → 7: Class weight and bias have been added to overcome the data imbalance problem (see Section 5.1). This lead to an improvement in 2 metrics (F1-score and accuracy) out of three while only slightly impacting the AUC.

7 → 8: Although the choice of Adam optimizer does not seem justified by the metrics values computed on the test set (especially F1-score and accuracy), training graphs are critical for this choice. Indeed, Figure 6.18 displays the training and validation metrics. The benefit of Adam is thus more visible, even though both experiments give quite chaotic curves.




(a) SGD optimizer.




(b) Adam optimizer.

Figure 6.18: Learning curves for different optimizers.

8 → 9: In addition to the class weights and bias, a sampling process has been implemented to address the imbalance issue. This time, the benefit is quite significant, as demonstrated by all the metric values.

9 → 10: Once more, some experiments concerning the batch size have been conducted. However, this time, batches of 256 seem to work better than smaller ones. 256 is thus the chosen final size.

10 → 11: Finally, the combination of hyperparameters of experiment 10 has been used to train a larger dataset, as supervised learning is supposed to benefit from large training sets. Since the two final tests yield the best results, the corresponding hyperparameters have been retained to conduct further experiments while decreasing the size of the training set. The largest dataset has been kept to do so.

### Visualization

Some visualization tools allow showing where the network focuses when it takes its decision. One of these visualization techniques, named **Grad-CAM** (Gradient-weighted Class Activation Mapping), has been applied to the above experiment 11 (see Table 6.3). The underlying principle of Grad-CAM is based on the fact that the last convolutional layer of the network contains high-level class-specific information. Therefore, the algorithm uses it to construct a localization map where the regions that trigger the final decision are highlighted. More precisely, «Grad-CAM uses the gradient information flowing into the last convolutional layer of the CNN to assign importance values to each neuron for a particular decision of interest» [74]. Although the expected purpose of this method is to support the fact that the model is efficient by showing it focuses on the right features, this visualization process is also a tool to detect issues in the learning process. Indeed, sometimes, networks can differentiate a class from another but not based on the desired features of the images.

Grad-CAM has been applied to four images, two with PEs and two without, and the resulting localization maps prove that although experiment 11 seems to be the best so far, the network still has a hard time accomplishing its desired task. Figure 6.19 shows the location of the potential pulmonary embolisms, and Figure 6.20 shows where the network focuses when it takes its decision.

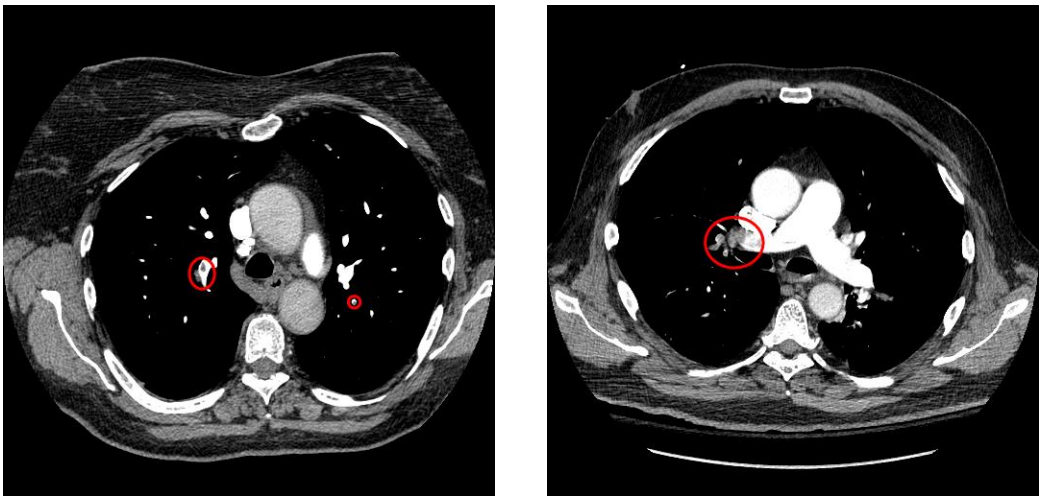
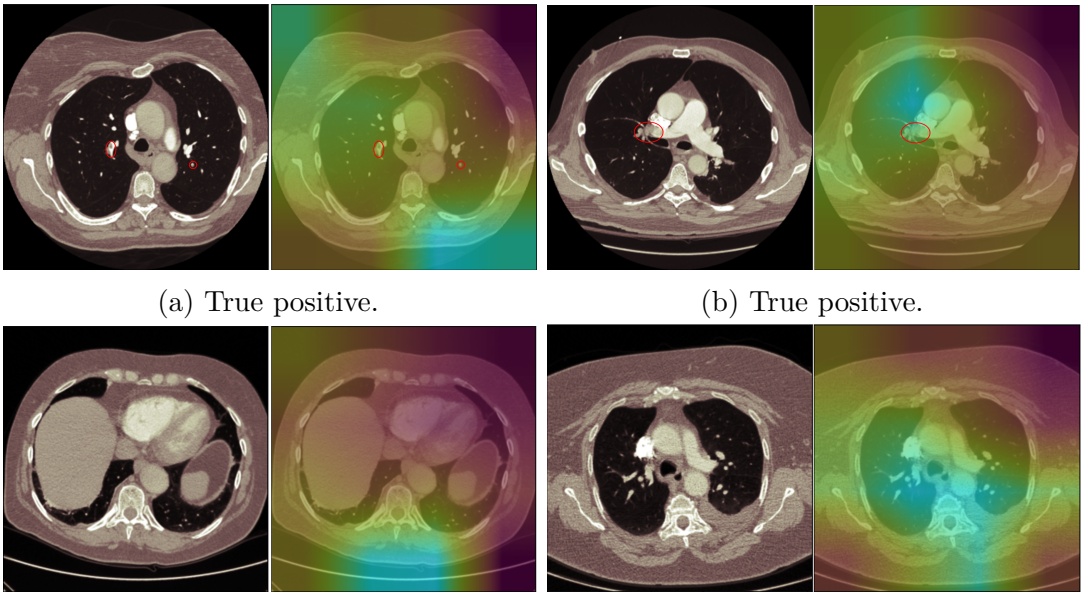


Figure 6.19: Pulmonary embolism localization on scans from two patients. The red circles are the locations of potential PEs.

**NB**: The localization of PEs is approximate. It has been performed only for the images alone with no verification with the entire set of scans covering the whole lung volume. Therefore the level of certainty is not maximal. However, this corresponds

to the task of the network (i.e., classification of 2D images without having access to the whole lung volume). In addition, although the localization is not sure, the circled areas of Figure 6.19 still are potential abnormalities.



(a) True positive. (b) True positive.

(c) False positive - patient with PE. (d) False positive - healthy patient.

Figure 6.20: Grad-CAM visualization from experiment 11 on scans from 4 different patients.

In this case, Grad-CAM supports the fact that the network is not efficient yet. Indeed, for the cases illustrated in Figures 6.20a and 6.20b, even though the classification is correct, it is not based on the detection of pulmonary embolisms (i.e., the network does not focus on the right part of the scan). In Figure 6.20a, while the PEs are situated in the left and right lungs (according to the dataset annotation and the above localization), the most intense area (i.e., blue area) is not in the lungs. As for Figure 6.20b, although nearer, the decision spot is too high compared to the PE (which is, in this case, noticeable). In addition, there are still many false positive predictions for both patients with and without pulmonary embolisms. Those also prove that the network bases its decision on the wrong elements.

For now, the above hyperparameters tuning process has mainly served to obtain the least bad model. However, for this work (i.e., the study of ways to overcome the annotation issue), further tests have been performed while decreasing the size of the training set. Therefore, the choice of the above hyperparameters still serves as a basis for those experiments.

### 6.3.2 Methods comparison

The second step of this task is to decrease the training set size to see how possible it is to reduce the annotation work. The aim is to quantify the impact of contrastive learning when networks only have few labeled images to train. For all those experiments, neither the validation set nor the test set has been modified.

As explained in Subsection 6.2.1, the training process has a random character that can significantly impact the results. Indeed, even after setting up the training to be the most reproducible possible, results are still considerably varying. Therefore, each supervised experiment has been run three times with the same setup. However, for the sake of time, only two contrastive experiments have been performed for each training set. The means of those multiple versions are the values that appear in Table 6.4.

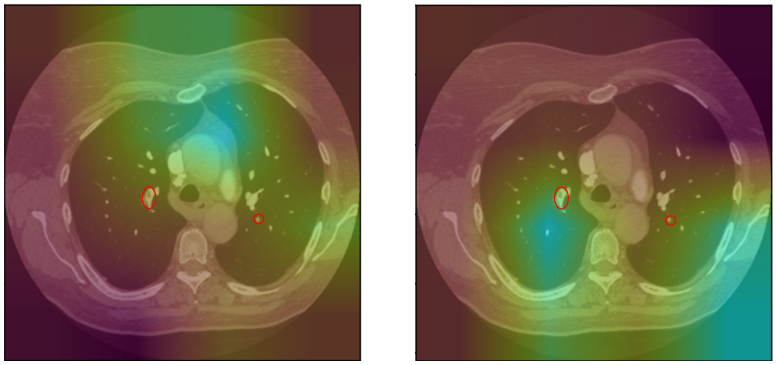
| | Training set percentage [%] | 100 | 90 | 80 | 75 | 66.66 | 50 | 33.34 | 25 | 20 | 10 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AUC | Supervised | 0.862 | 0.823 | 0.8613 | 0.833 | 0.855 | 0.863 | 0.879 | 0.888 | **0.902** | 0.87 | 0.817 |
| | Contrastive | 0.824 | 0.895 | 0.815 | 0.819 | **0.918** | 0.895 | 0.88 | 0.847 | 0.876 | 0.873 | 0.779 |
| F1-score | Supervised | 0.721 | 0.707 | 0.74 | 0.695 | 0.719 | 0.**764** | 0.755 | 0.738 | 0.**764** | 0.753 | 0.711 |
| | Contrastive | 0.689 | 0.76 | 0.755 | 0.718 | **0.805** | 0.785 | 0.741 | 0.702 | 0.746 | 0.742 | 0.644 |
| Accuracy | Supervised | 0.819 | 0.774 | **0.829** | 0.776 | 0.791 | 0.826 | 0.824 | 0.796 | 0.822 | 0.798 | 0.769 |
| | Contrastive | 0.77 | 0.819 | 0.845 | 0.755 | **0.863** | 0.853 | 0.801 | 0.782 | 0.808 | 0.783 | 0.654 |

Table 6.4: Results of Cases 2 and 4. The bold values are the best of their line.

The first conclusion drawn from Table 6.4 joins the one from the previous subsection, that is, the learning process does not seem to be efficient. Indeed, the networks achieve the same range of results for almost all percentages of the training set. Sometimes results are even better with ten times fewer images. Therefore, the performance with a large part of the training set should be improvable.
Secondly, contrastive pre-training has no impact on model performance. Indeed, the results vary so much between both methods that it is hard to know if it is due to the method, or the randomness of the simulation, or the fact the training is not efficient yet.

### Visualization

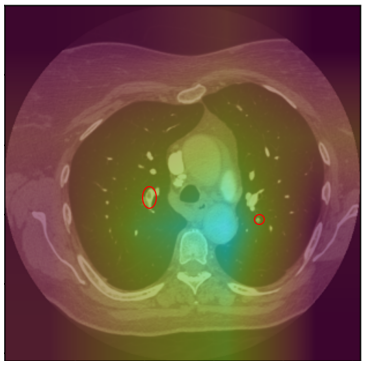
Concerning the source of the networks' decisions, once again, they are not relevant. In addition, no visible improvements are present with contrastive learning.
Grad-CAM has been applied on both images from Figure 6.19 for two different tests: with 90 and 10% of the training set. The results are depicted in Figures 6.21 and 6.22.



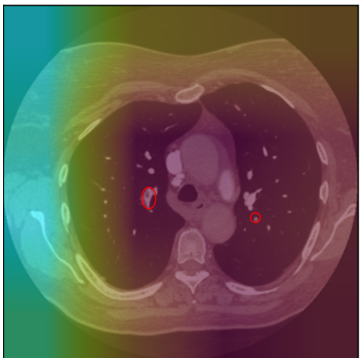
(a) Supervised: 90%.



(b) Supervised: 10%.



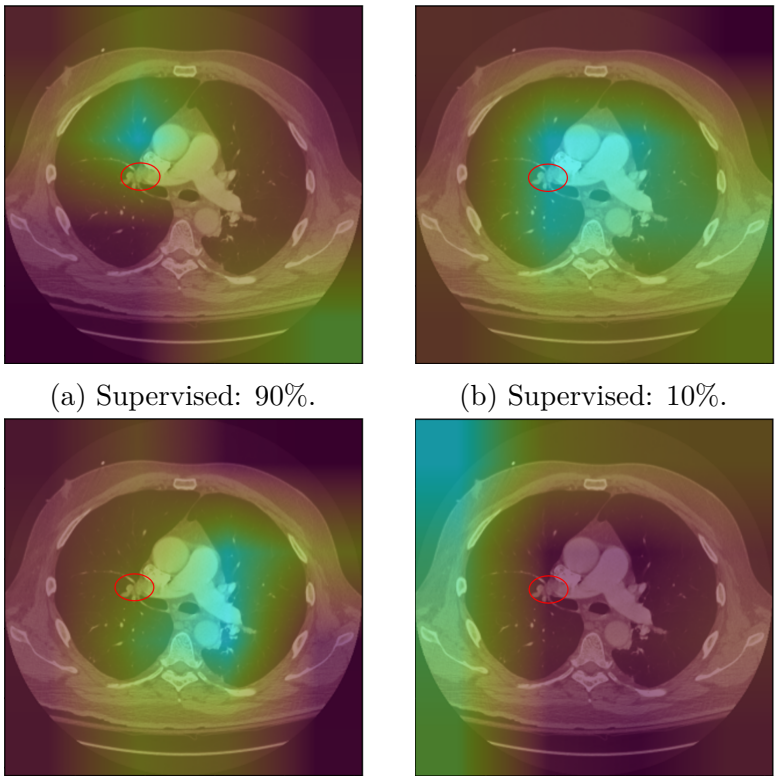
(c) Contrastive: 90%.



(d) Contrastive: 10%.

Figure 6.21: Grad-CAM visualization from supervised and contrastive experiments with 90 and 10% of training set: first image.

For this first image, although all the networks classify it in the right category, none of them seem to base their decision on reliable features. Moreover, there does not seem to be a common source that mistakes the network. Indeed, they are all right in their prediction while being wrong in different ways about their decision criteria.
Concerning the second image (Figure 6.22), the results are not convincing either. Although one experiment (Figure 6.22b) comprises the PE in its focus area, the level of precision is weak as this area is widely spread in the center of the scan.



(a) Supervised: 90%. (b) Supervised: 10%.

(c) Contrastive: 90%. (d) Contrastive: 10%.

Figure 6.22: Grad-CAM visualization from supervised and contrastive experiments with 90 and 10% of training set: second image.

From then on, the results lead to the fact that in any case, before efficiently comparing both methods, the learning process should be rethought for models to achieve their task. Only then, the comparison with another method will make sense.

### 6.3.3 Areas for improvement

Designing an efficient deep network training is a complex and meticulous task. Some of the following ideas could therefore help better understand and improve the experiment.

The complexity of the learning process is dependent on the dataset. Therefore, firstly, a deeper study of the dataset could be of great interest. Indeed, the dataset from RSNA contains multiple forms of pulmonary embolisms, which might be confusing for the learning process. Therefore, it could be more effective to select

with more attention which images should be part of the training, validation, and test sets.
In addition, regarding the selection of patients for the test set, statistical analysis could also help choose «challenging» images and not a set highly correlated with another used for training.

Secondly, an efficient implementation takes time to build. Throughout this thesis, one major problem was the random aspect of the simulations. Therefore, it could be beneficial to take more time for each step of the process. For example, when tuning the hyperparameters, several tests with the same configuration and a cross-validation scheme (see Subsection 6.2.1) could help make relevant choices. Indeed, it would bring a deeper understanding of the impact of each modification.

Thirdly, concerning this particular dataset and task, simple training with a CNN might be inadequate. This dataset has been used in a Kaggle challenge [75]. Although the purpose of this one was more complex (classification at image and study levels), multiple teams have implemented more complex structures to perform the classification. For example, many participants have performed the task in several stages: a CNN for features detection and more complex implementations for the classification part. Secondly, some of the teams used the input image but also its neighbors in the learning process. Finally, in terms of image preprocessing, sometimes lungs localization was performed. In addition, some of the teams kept the full image resolution to avoid losing relevant information [76, 75]. Therefore, an implementation closer to what was done in this challenge might be more suited for the second task of this thesis.

Finally, regarding contrastive learning, even for end-to-end mechanisms (see Subsection 3.1.1), larger batches are used, and networks are trained longer [43]. Therefore, it might be interesting to perform more varied contrastive pre-trainings to quantify the impact of those hyperparameters.
The fine-tuning process is also a critical step and is worth studying further. Indeed, it is this stage that reflects the performance of contrastive pre-training. A wrong setting could lead to a waste of the pre-training potential. Once again, time should be taken to try different methods (one-step /two-step fine-tuning, transfer learning, etc.) to evaluate which one is the most suited for the downstream task.

## 6.4 Discussion

### 6.4.1 Deep learning for pulmonary embolism

As said in Subsection 6.3.3, the design of an efficient deep network is a challenging task. Moreover, the detection of pulmonary embolisms is another. Indeed, the diagnosis of PE can be disrupted by the presence of many PE look-alikes such as multiple kinds of artifacts [77] (respiratory motion, streak artifacts [78], flow-related artifacts [79], etc.), lymph nodes, vascular bifurcation, etc. [57]. Those are the principal cause of false-positive predictions. Therefore, creating an end-to-end process to detect pulmonary embolisms with precision is a critical and delicate task.

Up to now, about PE diagnosis and detection, AI has mainly been used for Computer-Aided Detection (CAD) [80], using traditional machine learning techniques [81]. However, they were suffering from a high false positive rate and were not easily reviewed quickly [82, 83].
It is in 2015 that the first deep learning method was used for PE detection [83]. This first application of deep learning already achieved satisfying results outperforming the previous ones.
Since then, few other methods have been developed. Those are still mainly in the research phase and need to be deployed to determine their potential impact [81]. Deep learning for PE detection is therefore a growing topic that looks promising.

### 6.4.2 Future work

Now that PE detection has been introduced to deep learning, this last can be used for multiple tasks to meet the demand of medical experts.
Other tasks such as artery and vein segmentation and cardiac chamber segmentation may be helpful and relevant for the PE diagnosis. Indeed, as PEs are mainly located in the arteries, a pre-segmentation could optimize the next detection step. About the heart chambers, their segmentation would aim to compare their volumes. Indeed, abnormal dilatation of the right chambers is one of the prognostic factors of PE [84].

In line with this thesis, the next topic to investigate could therefore be segmentation. Moreover, a transition from 2D to 3D would be beneficial to maximize the performance of deep networks for this task. For those applications, contrastive learning should also be studied as it still shows promising results, despite those in Case4.
Regarding the segmentation, [85] already proposes a local extension of the contrastive loss more suited for tasks as image segmentation. In addition, as the

annotation work is heavier in segmentation tasks, the process of annotation enrichment is worth being studied.
About the transition to 3D, contrastive learning would also need to be adapted. Indeed, 3D inputs introduce way more parameters than 2D and occupy more memory. However, end-to-end contrastive learning also requires a lot of memory as it benefits from large batches. Therefore, for a 3D application, other mechanisms like the momentum encoder (see Subsection 3.1.1) would be more suited and deserved to be studied.

Finally, shortly, a dataset from «Cliniques universitaires Saint-Luc» should be available for private research purposes. This dataset has the particularity of containing two sources of information. In addition to standard CT scans, iodinated contrast scans are also available. Both those scans are complementary as PEs are sometimes more visible on one than on the other. Therefore, these iodinated contrast scans should be part of future works as they bring extra information that could help detect PE.

CHAPTER 7

# Conclusion

This thesis is part of the research about the usage of deep learning for pulmonary embolism diagnosis and detection. However, the most widely used deep learning techniques (i.e., supervised techniques) require a large dataset accurately annotated, which is a limitation, particularly for the medical world. This problem has inspired this work to focus on two main questions: how to alleviate the annotation process and how to learn with few data.
The first aspect is treated through an annotation enrichment task, while the second question led to the implementation of contrastive learning.

The study of contrastive learning has been conducted through two different tasks, including annotation enrichment. The other aims to develop a model to perform a general binary classification between images with and without pulmonary embolism. This method is of great interest since previous researches prove that it can achieve better results than supervised learning when training with fewer data.
Although the benefit of contrastive learning has not been put forward with this second task, an improvement is observed in the first task.

Concerning the annotation enrichment, a ResNet50 has been implemented to perform the same binary classification. The network is trained with several fractions of the training set to predict the annotation of the remaining images. To ensure the smooth running of the process, samples that must be annotated belong to the same patients with which the network is trained. Finally, test-time augmentation is used to estimate the uncertainty of the model.
With supervised training, the annotation work can be considerably reduced. Only 52.95% of the dataset needs to be labeled. Indeed, by training with only a third of the dataset, the network can learn enough to annotate the remaining images with

a certainty of 81.09% and a perfect accuracy. These are theoretical results and not practical ones since they are obtained considering the test set labels.
These results are improved when implementing the contrastive pre-training. With this last one followed by a downstream fine-tuning step, the final percentage of images to be annotated is reduced to 49.34%. In addition, it also reduces the training costs as only 25% of the dataset is used for the training stage.

The usage of deep learning for pulmonary embolism detection is still in its early days, and contrastive methods are increasingly studied due to their promising results. Therefore, there is a vast future to explore concerning the use of artificial intelligence for medical purposes.

# Bibliography


[1] E.-O. Essien, P. Rali, and S. C. Mathai. Pulmonary Embolism. *Medical Clinics of North America*, 103(3):549–564, May 2019.

[2] M. V. Huisman, S. Barco, S. C. Cannegieter, et al. Pulmonary embolism. *Nature Reviews Disease Primers*, 4(1):10828, 2018.

[3] J. Hirsh, R. D. Hull, and G. E. Raskob. Diagnosis of pulmonary embolism. *Journal of the American College of Cardiology*, 8(6, Supplement 2):128B–136B, 1986. Symposium on Thrombosis and Antithrombotic Therapy—1986.

[4] J. A. Soye, C. B. Loughrey, and P. D. Hanley. Computed Tomography Pulmonary Angiography: A Sample of Experience at a District General Hospital. *The Ulster Medical Journal*, 77(3):175–180, Sep 2008.

[5] S.-C. Huang, T. Kothari, I. Banerjee, et al. PENet—a scalable deep-learning model for automated diagnosis of pulmonary embolism using volumetric CT imaging. *npj Digital Medicine*, 3:61, Dec 2020.

[6] E. Colak, F. C. Kitamura, S. B. Hobbs, C. C. Wu, et al. The RSNA Pulmonary Embolism CT Dataset. *Radiology: Artificial Intelligence*, 3(2):e200254, 2021.

[7] Z. M. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani. State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow's Intelligent Network Traffic Control Systems. *IEEE Communications Surveys & Tutorials*, 19(4):2432–2455, 2017.

[8] A. Shrestha and A. Mahmood. Review of Deep Learning Algorithms and Architectures. *IEEE Access*, 7:53040–53065, 2019.

[9] J. Schmidhuber. Deep Learning. *Scholarpedia*, 10(11):32832, 2015.

[10] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis. Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*, 2018(ID 7068349):1–13, 2018.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, 60(6):84–90, May 2017.

[12] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, et al. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, et al. ImageNet Large Scale Visual Recognition Challenge. *arXiv preprint arXiv:1409.0575*, Jan 2015.

[14] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*, Apr 2014.

[15] C. Szegedy, W. Liu, J. Jia, P. Sermanet, et al. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Boston, MA, USA, Jun 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, Jun 2016.

[17] H. Wu, Q. Liu, and X. Liu. A Review on Deep Learning Approaches to Image Classification and Object Segmentation. *Computers, Materials Continua*, 60(2):575–597, 2019.

[18] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, Antalya, Aug 2017.

[19] S. Walczak. Artificial Neural Networks. In *Advanced Methodologies and Technologies in Artificial Intelligence, Computer Simulation, and Human-Computer Interaction*, pages 40–53. IGI Global, 2019.

[20] M. Verleysen. Lecture notes in Machine Learning : regression, dimensionality reduction and data visualization, November 2019. Université Catholique de Louvain.

[21] S.-C. Wang. Artificial Neural Network. In *Interdisciplinary Computing in Java Programming*, pages 81–100, Boston, MA, 2003. Springer US.

[22] A.T.C. Goh. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151, 1995.

[23] R. Hecht-Nielsen. *Theory of the Backpropagation Neural Network*, page 65–93. Harcourt Brace Co., USA, 1992.

[24] E. Brion. Deep learning for organ segmentation in radiotherapy : federated learning, contour propagation, and domain adaptation. Prom. : Lee, John ; Macq, Benoit `http://hdl.handle.net/2078.1/244327`, 2021.

[25] F. Taherkhani, J. Dawson, and N. M. Nasrabadi. Deep Sparse Band Selection for Hyperspectral Face Recognition. *arXiv preprint arXiv:1908.09630*, Aug 2019.

[26] C. Wang. Lecture notes in Deep Learning For Medical Images Analysis, Sep 2020. KTH Royal Institute of Technology.

[27] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten Digit Recognition with a Back-Propagation Network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 396–404. Morgan-Kaufmann, 1990.

[28] J. A. Lee. Lecture notes in Machine Learning : regression, dimensionality reduction and data visualization, November 2019. Université Catholique de Louvain.

[29] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[31] Q. Ji, J. Huang, W. He, and Y. Sun. Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images. *Algorithms*, 12(3):51, Feb 2019.

[32] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 2011. JMLR Workshop and Conference Proceedings.

[33] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013.

[34] S. Azizi, B. Mustafa, F. Ryan, Z. Beaver, et al. Big Self-Supervised Models Advance Medical Image Classification. *arXiv preprint arXiv:2101.05224*, Apr 2021.

[35] A. Kolesnikov, X. Zhai, and L. Beyer. Revisiting Self-Supervised Visual Representation Learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1920–1929, Long Beach, CA, USA, 2019.

[36] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon. A Survey on Contrastive Self-supervised Learning. *arXiv preprint arXiv:2011.00362*, 2020.

[37] R. Zhang, P. Isola, and A. A. Efros. Colorful Image Colorization. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision - ECCV 2016*, volume 9907, pages 649–666, Cham, 2016. Springer International Publishing.

[38] J.-B. Grill, F. Strub, F. Altché, C. Tallec, et al. Bootstrap your own latent: A new approach to self-supervised Learning. *arXiv preprint arXiv:2006.07733*, 2020.

[39] P. H. Le-Khac, G. Healy, and A. F. Smeaton. Contrastive Representation Learning: A Framework and Review. *IEEE Access*, 8:193907–193934, 2020.

[40] S. Gidaris, P. Singh, and N. Komodakis. Unsupervised Representation Learning by Predicting Image Rotations. *arXiv preprint arXiv:1803.07728*, 2018.

[41] M. Noroozi and P. Favaro. Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles. *arXiv preprint arXiv:1603.09246*, 2017.

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

[43] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *arXiv preprint arXiv:2002.05709*, Jun 2020.

[44] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. *arXiv preprint arXiv:1911.05722*, 2020.

[45] A. van den Oord, Y. Li, and O. Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv preprint arXiv:1807.03748*, 2018.

[46] Z. Wu, Y. Xiong, S. Yu, and D. Lin. Unsupervised Feature Learning via Non-Parametric Instance-level Discrimination. *arXiv preprint arXiv:1805.01978*, 2018.

[47] J. Xie, X. Zhan, Z. Liu, Y. S. Ong, and C. C. Loy. Delving into Inter-Image Invariance for Unsupervised Visual Representations. *arXiv preprint arXiv:2008.11702*, 2021.

[48] I. Misra and L. van der Maaten. Self-Supervised Learning of Pretext-Invariant Representations. *arXiv preprint arXiv:1912.01991*, 2019.

[49] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 2010. JMLR Workshop and Conference Proceedings.

[50] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the Limits of Language Modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[51] F. Wang and H. Liu. Understanding the Behaviour of Contrastive Loss. *arXiv preprint arXiv:2012.09740*, 2021.

[52] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. *arXiv preprint arXiv:2103.03230*, 2021.

[53] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[54] M. Remy-Jardin, J.-B. Faivre, R. Kaergel, A. Hutt, et al. Machine Learning and Deep Neural Network Applications in the Thorax. *Journal of Thoracic Imaging*, 35:S40–S48, May 2020.

[55] J. Ma, Y. Song, X. Tian, Y. Hua, R. Zhang, and J. Wu. Survey on deep learning for pulmonary medical imaging. *Frontiers of Medicine*, 14(4):450–469, Aug 2020.

[56] Fundamentals Of Computed Tomography Studies: Windowing. *Stepwards*. `https://www.stepwards.com/?page_id=21646`. Online; accessed April 26 2021.

[57] C. Wittram, M. M. Maher, A. J. Yoo, M. K. Kalra, J.-A. O. Shepard, and T. C. McLoud. CT Angiography of Pulmonary Embolism: Diagnostic Criteria and Causes of Misdiagnosis. *RadioGraphics*, 24(5):1219–1238, 2004.

[58] G. Wang, W. Li, M. Aertsen, J. Deprest, S. Ourselin, and T. Vercauteren. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. *Neurocomputing*, 338:34–45, Apr 2019.

[59] M. S. Ayhan and P. Berens. Test-time Data Augmentation for Estimation of Heteroscedastic Aleatoric Uncertainty in Deep Neural Networks. In *International conference on Medical Imaging with Deep Learning*, Amsterdam, 2018.

[60] I. Kandel and M. Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, 2020.

[61] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal Loss for Dense Object Detection. *arXiv preprint arXiv:1708.02002*, 2018.

[62] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[63] H. Yu, R. Jin, and S. Yang. On the Linear Speedup Analysis of Communication Efficient Momentum SGD for Distributed Non-Convex Optimization. *arXiv preprint arXiv:1905.03817*, 2019.

[64] G. Leclerc and A. Madry. The Two Regimes of Deep Network Training. *arXiv preprint arXiv:2002.10376*, 2020.

[65] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015.

[66] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, Jan 2014.

[67] Y. Li, C. Wei, and T. Ma. Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[68] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2017.

[69] J.H. Jeppesen, R. H. Jacobsen, F. Inceoglu, and T. S. Toftegaard. A cloud detection algorithm for satellite imagery based on deep learning. *Remote Sensing of Environment*, 229:247–259, 2019.

[70] J. M. Cochran. Diffuse Optical Biomarkers Of Breast Cancer. 2018. Dissertation, Degree of Doctor of Philosophy, University of Pennsylvania.

[71] D. Berrar. Cross-Validation. In S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach, editors, *Encyclopedia of Bioinformatics and Computational Biology*, pages 542–545. 2019.

[72] L. van der Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[73] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G. Hinton. Big Self-Supervised Models are Strong Semi-Supervised Learners. *arXiv preprint arXiv:200610029*, 2020.

[74] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019.

[75] RSNA STR Pulmonary Embolism Detection. *Kaggle*. `https://www.kaggle.com/c/rsna-str-pulmonary-embolism-detection/overview`. Online; accessed August 11 2021.

[76] RSNA Pulmonary Embolism Detection Challenge (2020). *RSNA*. `https://www.rsna.org/education/ai-resources-and-training/ai-image-challenge/rsna-pe-detection-challenge-2020?fbclid=IwAR0-bRn7iC0ETy_tYmojups1UvrIqthL9U_hWoj0l_dqiTMoclUT79kd2ss`. Online; accessed August 11 2021.

[77] P.T. Johnson. *Artifacts mimicking pulmonary embolism*, page 134–136. Cambridge University Press, 2015.

[78] A.-M. Sykes. *Streak artifacts*, page 217–217. Cambridge University Press, Cambridge, 2011.

[79] K. Stefanidis, J. Green, E. Konstantelou, and H. Robbie. Flow artefact mimicking pulmonary embolism in pulmonary hypertension. *BMJ Case Reports CP*, 13(2), 2020.

[80] R. A. Castellino. Computer aided detection (CAD): an overview. *Cancer Imaging*, 5(1):17–19, 2005.

[81] S. Soffer, E. Klang, O. Shimon, Y. Barash, N. Cahan, H. Greenspana, and E. Konen. Deep learning for pulmonary embolism detection on computed tomography pulmonary angiogram: a systematic review and meta-analysis. *Scientific Reports*, 11(1):15814, Dec 2021.

[82] N. Tajbakhsh, J. Y. Shin, M. B. Gotway, and J. Liang. Computer-aided detection and visualization of pulmonary embolism using a novel, compact, and discriminative image representation. *Medical Image Analysis*, 58:101541, 2019.

[83] N. Tajbakhsh, M. B. Gotway, and J. Liang. Computer-Aided Pulmonary Embolism Detection Using a Novel Vessel-Aligned Multi-planar Image Representation and Convolutional Neural Networks. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 62–69, Cham, 2015. Springer International Publishing.

[84] S. Carerj, M. P. Trifiro, C. Zito, S. de Salvo, and F. Consolo. [cardiologic diagnosis of pulmonary embolism: echocardiography]. *Minerva Cardioangiologica*, 48(12 Suppl 1):15–20, Dec 2000.

[85] K. Chaitanya, E. Erdil, N. Karani, and E. Konukoglu. Contrastive learning of global and local features for medical image segmentation with limited annotations. *arXiv preprint arXiv:2006.10511*, 2020.

# Appendices

APPENDIX A

# Learning rate scheduling and overfitting

The scheduling of the learning rate can be a decisive matter to plan. Indeed, it has a considerable impact on the training process. A common practice concerning the learning rate is to decrease it throughout the process to refine the weights update as the network converges. Therefore, a scheduled decay is applied to the learning rate according to a particular function (step, exponential, etc.). This one can be dependant either on the epochs or on the iterations. For example, for a stepwise decay, the learning rate could drop after several epochs or iterations.

As a reminder, an epoch consists of one complete cycle through the training set. Then, just as the training set is divided into batches, each epoch is divided into iterations. These correspond thus to one pass through the network. Hence, the number of iterations per epoch corresponds to the number of batches for the whole dataset. Therefore, when decreasing the size of the training set, the epochs contain fewer iterations. So, when using a function that changes the learning rate according to the epochs, the decay schedule becomes dependent on the size of the training set.

When performing one experiment, both types of scheduling can be designed to be equivalent. Therefore, they can both be used without troubling the process. However, for the kind of procedure found in this thesis (i.e., series of tests with a decreasing training set), the choice of decay highly impacts the learning stage. Indeed, wrong scheduling can increase the risk of overfitting. The following example illustrates this interference between the chosen scheduler and the problem of overfitting.

During the experimental phase of this master thesis, numerous tests were made to find the optimal set of hyperparameters. For Case2 (i.e., development of a generalizable model), an epoch-based step decayed learning rate (Figure A.1) was first implemented.
However, as the training set was reduced, a strange phenomenon has been observed. When using the entire set, the learning process was suffering from overfitting, whereas when using, for example, 25% of the training data, the learning curves were better. Figure A.2 depicts this observation.

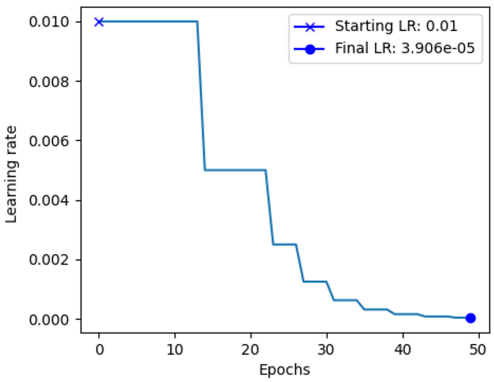
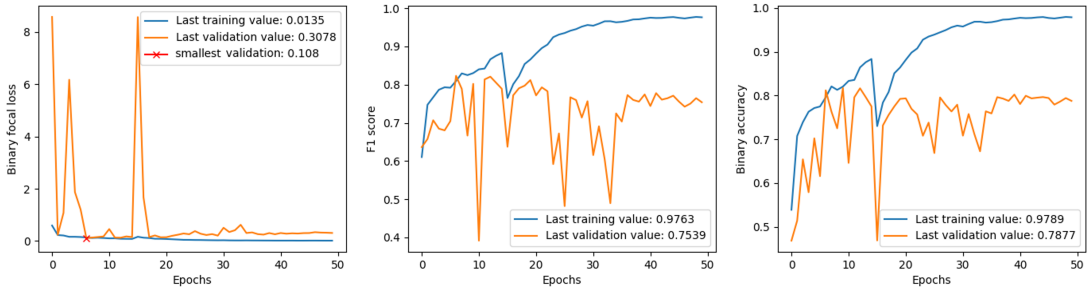
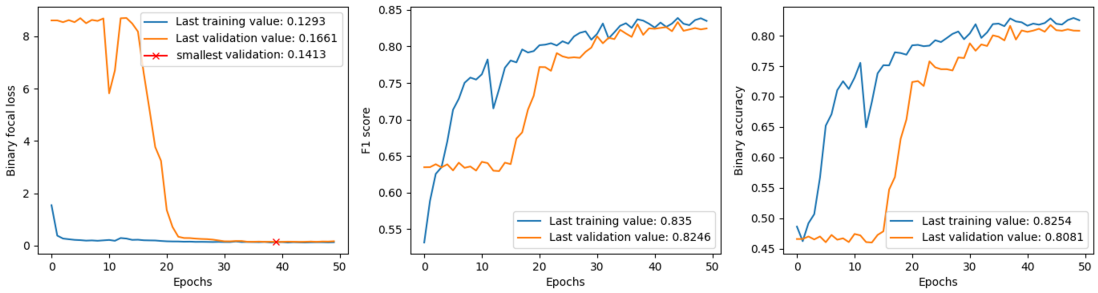



Figure A.1: Epoch-based step decayed learning rate.




(a) 100% of training set (i.e., 16299 images).




(b) 25% of training set (i.e., 4088 images).

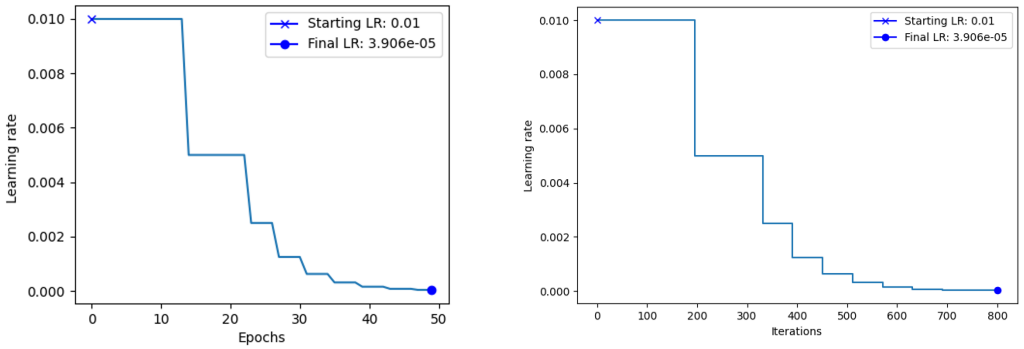
Figure A.2: Learning curves for different training set sizes and an epoch-based learning rate schedule.

**NB**: The smallest validation value is presented for information only. Indeed, the models kept for the test phase are always the ones after all the epochs. No

EarlyStopping (see Subsection 4.1.2) has been performed in this work.

This kind of behavior is unexpected as supervised learning is supposed to benefit from large databases. The underlying assumption about this training improvement is that the chosen learning rate scheduling is more suitable when using a small dataset. Indeed, when the decay depends on the epochs, the first drop occurs later when using the entire training set than when using 25% of it. Therefore, the whole decay scheme is more spread out during training, visibly increasing the risk of overfitting.

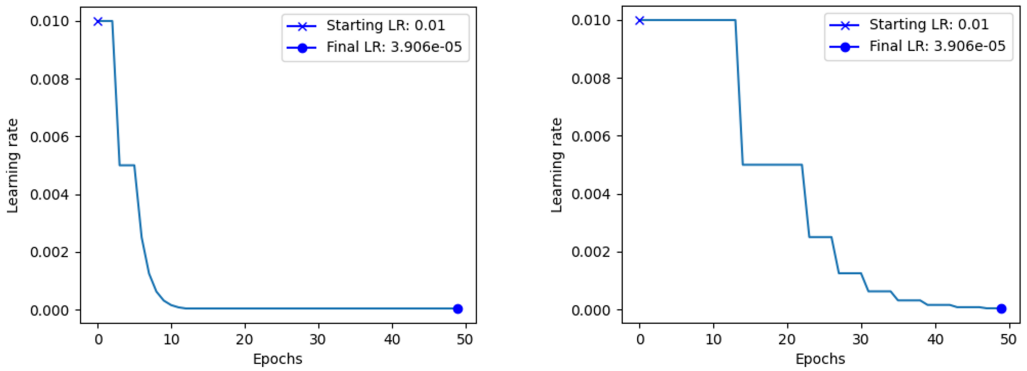
Other tests have been performed to fix this assumption, but with an iteration-based schedule, which does not depend on the training set size. As the decay choice of the experiment with 25% of the training set seemed well-suited, this epoch-based scheduling has been transformed into an iteration-based one. Indeed, knowing the batch size and the training set size, the number of iterations was computed, and each learning drop was associated with its specific iteration. Figure A.3 shows the transformation from epoch-based to iteration-based. In addition, Figure A.4 also shows the drops scheme according to the training set size.




(a) Epoch-based scheduling. (b) Equivalent iteration-based scheduling.

Figure A.3: Learning curves for different training set sizes and an epoch-based learning rate schedule.

Since the decay schedule no longer depends on the epochs, for a large training set, the learning rate drops earlier than before. This type of schedule seems to be better adapted to the learning process as it considerably decreases the gap between the training and validation curves (i.e., it suffers less from overfitting). This is visible in Figure A.5.




(a) 100% of the training set (i.e., 16299 images).

(b) 25% of the training set (i.e., 4088 images).

Figure A.4: Iteration-based learning rate schedule according to the training set size.

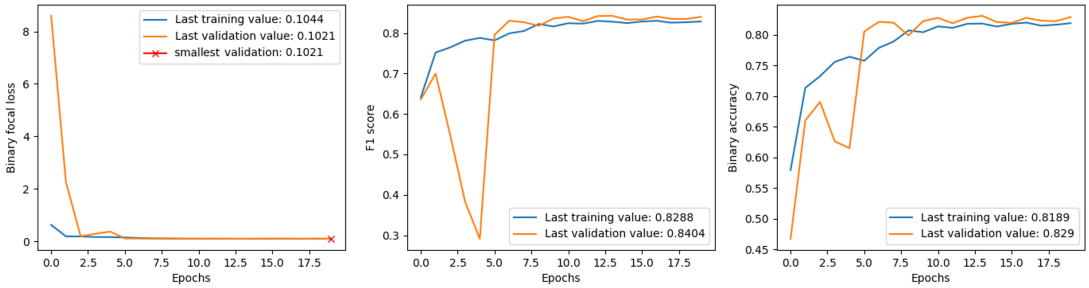
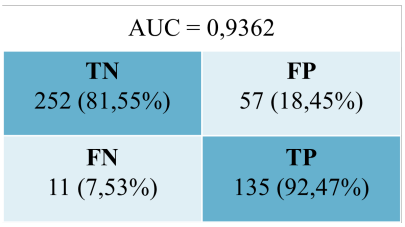


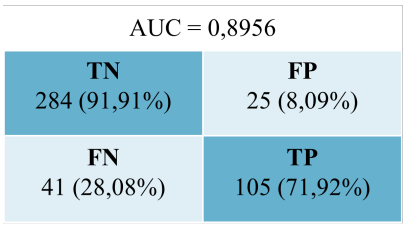

Figure A.5: Learning curves of a supervised experiment with the entire training set and the iteration-bases learning rate schedule.

However, although the curves are significantly improved, there is no improvement in the results on test sets. Indeed, Figure A.6 contains the confusion matrices and AUC of both tests illustrated in Figures A.2a and A.5. Those matrices have been computed for two different test sets: the first one with two patients and the second one with ten. However, as the classification task is not functional yet (see Section 6.3), the main take-home message is that there is a link between overfitting and the learning rate schedule.

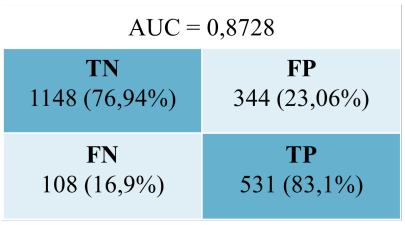
In conclusion, it is known that tuning the learning rate is not an easy task. It always depends on the experiment, even though some standard practices are recommended. Anyway, the above example highlights a direct link between the learning rate schedule and the problem of overfitting. Therefore, these experiments illustrate another risk people should be aware of when choosing a decay scheme for the learning rate.



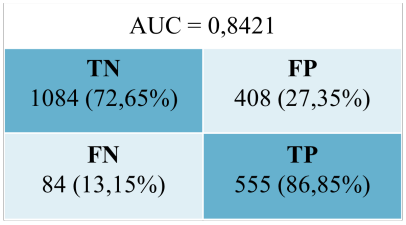

(a) Epoch-based training. Prediction on 2 patients.




(b) Iteration-based training. Prediction on 2 patients.




(c) Epoch-based training. Prediction on 10 patients.




(d) Iteration-based training. Prediction on 10 patients.

Figure A.6: Confusion matrices and AUC of networks trained with different learning rate scheduling.

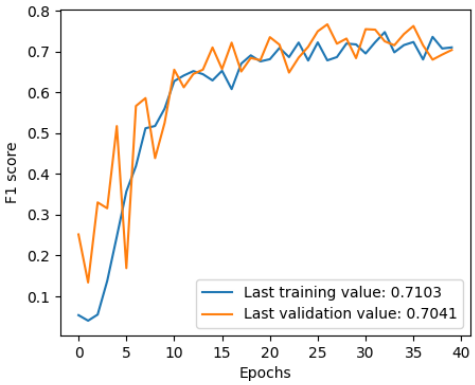
APPENDIX B

# F1-score interpretation

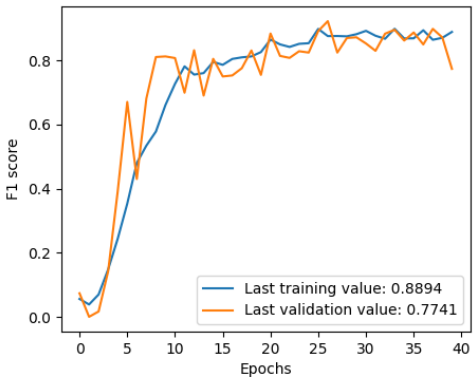
Although F1-score is supposed to be between 0 and 1, 1 being the best score, a higher score is not always a sign of better performance. This is illustrated by the following example. The same simulation (Test90 of Case1) has been performed three times with three different batch sizes: 32, 64, and 256. The results of those simulations are visible in Figure B.1.

When looking at the graphs, the models could be classified as follows, from the most to the less efficient: batch 64, batch 256, batch 32. However, when looking at the results when annotating the test set (Figure B.2), this classification does not seem to be correct.
This can be explained by looking at the distribution of positive images across the batches (see Subsection 6.2.2). Indeed, as the batch size grows, the probability of having batches without positive images decreases, and the higher bound of the F1-score increases. It reaches 1 for a distribution in which all batches contain at least one positive image. Therefore it is essential to look at this distribution to interpret the results.

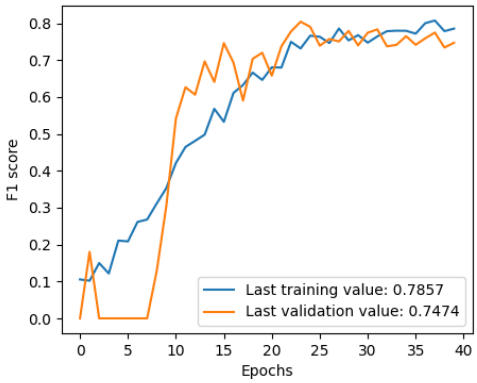
Figure B.3 shows the distribution of positive images respectively for one of the batches of sizes 32, 64, and 256.
When the batch size is 256, it always contains at least one positive image. The maximum F1-score for an epoch (and for the whole training) is therefore 1.
When the batch size is 64, a few batches of the epoch do not contain any positive images, hence impacting the maximum F1-score. In this case (Figure B.3b),the higher bound of F1-score for the epoch decreases to 0.9429 as there are ten batches without positive images (counting as 0 in the mean score) and 165 batches with at least one positive image (counting as 1 in the mean score). The mean higher F1-




(a) Batch size of 32.




(b) Batch size of 64.




(c) Batch size of 256.

Figure B.1: F1-score during training of Test90 of Case1.

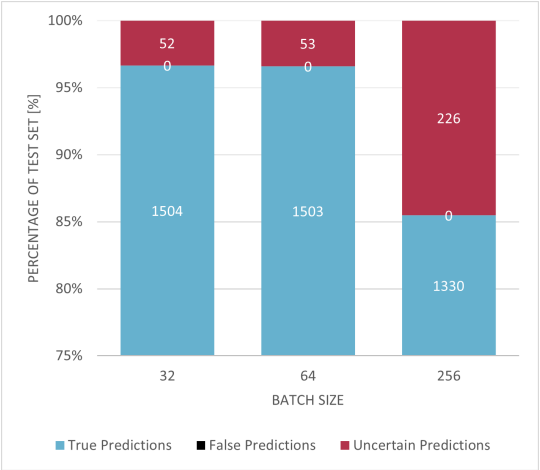
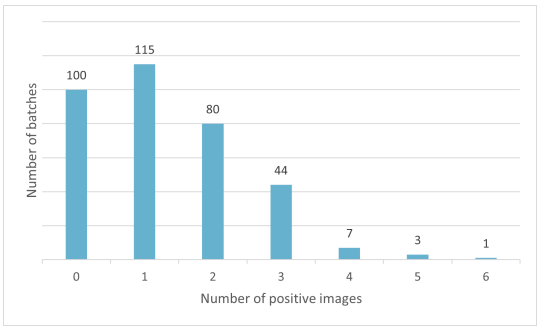
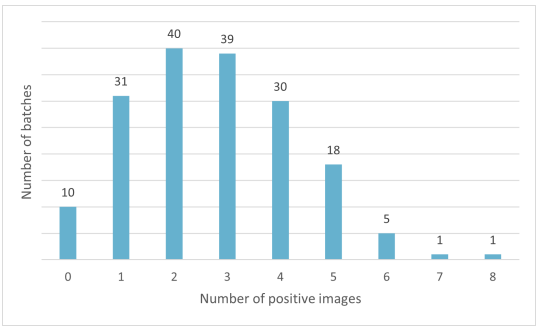


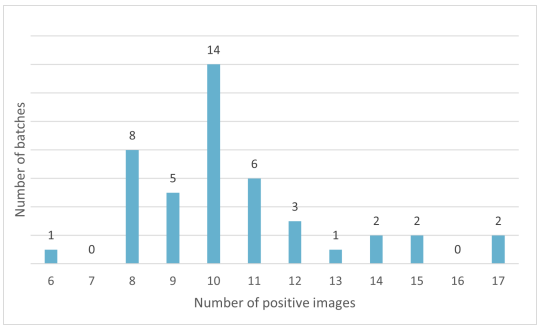

Figure B.2: Test-time augmentation results for thresholds of 0.75 and 0.25.




(a) Batch size of 32.




(b) Batch size of 64.




(c) Batch size of 256.

Figure B.3: Distribution of positive images across batches for one epoch.

score for the entire training (mean of all the epochs) becomes 0.93943. Eventually, when the batch size is 32, the higher bound for the epoch shown in Figure B.3a is 0.7143, and the mean maximal score is 0.7364 for the training.

The classification of the three networks can now be updated. The training with a batch size of 256 is the least efficient. Indeed, the gap between the validation and the maximal F1-score is the largest. It is followed by the training with batches of 64 images. Therefore, the network trained with a batch size of 32 is the best among the three. This new classification agrees with the test-time augmentation results display in Figure B.2.

The interpretation of a metric is a complicated task that can be easily misleading. Therefore, it is essential to go a step further and check the efficiency of the model for its final purpose, which is, in this case, the annotation enrichment.

APPENDIX C

# Additional results: Case2

## C.1 Impact of dataset distribution

Figures C.1 and C.2 show the different learning curves of experiments one and two from Table 6.3.

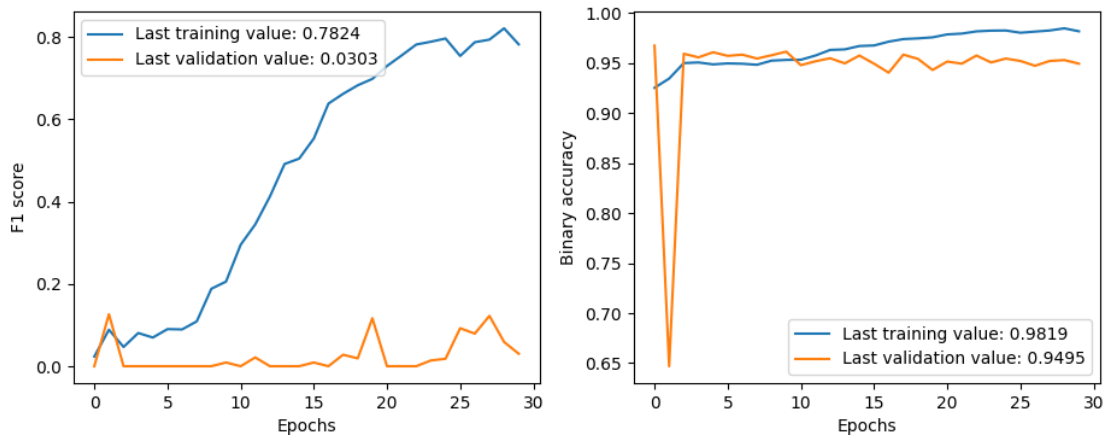



Figure C.1: Learning curves of experiment one from Table 6.3 with a dataset containing 4.21% of positive samples.

In Figure C.1, the network is trained on a dataset containing only 4.21% of positive samples. In that case, the learning process is highly impacted. Indeed, the F1-score that emphasizes the correct predictions of the positive class is almost null while the accuracy is above 94%. Those results suggest that the network rarely classifies an image rightly as positive.

Therefore, a second experiment with a dataset containing 16.09% of positive images has been conducted to soften the impact of the data imbalance. The learning curves displayed in Figure C.2 correspond to this second test. Even though the resulting metrics values and curves are not convincing yet, the impact of the new dataset is significant. Indeed, the F1-score increases considerably while the accuracy decreases. This suggests that the network, although making more mistakes, has more material to start classifying images into the two categories.

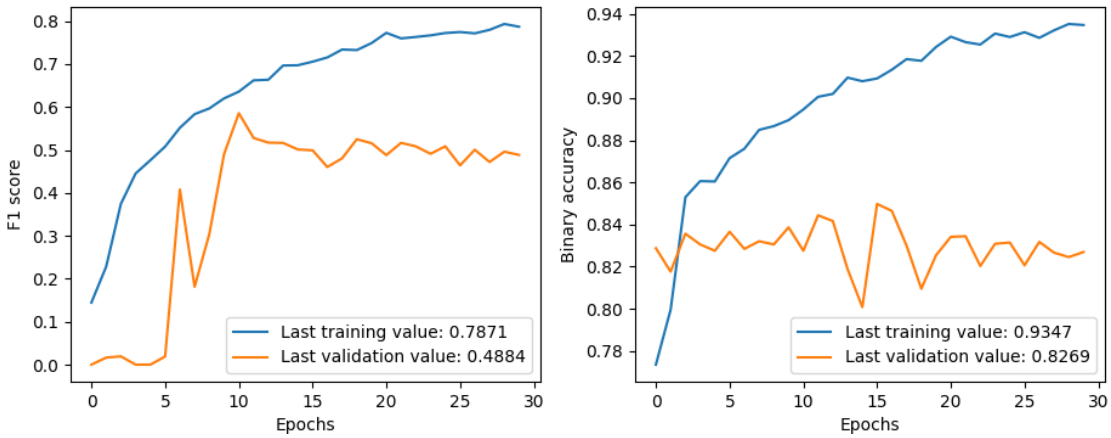
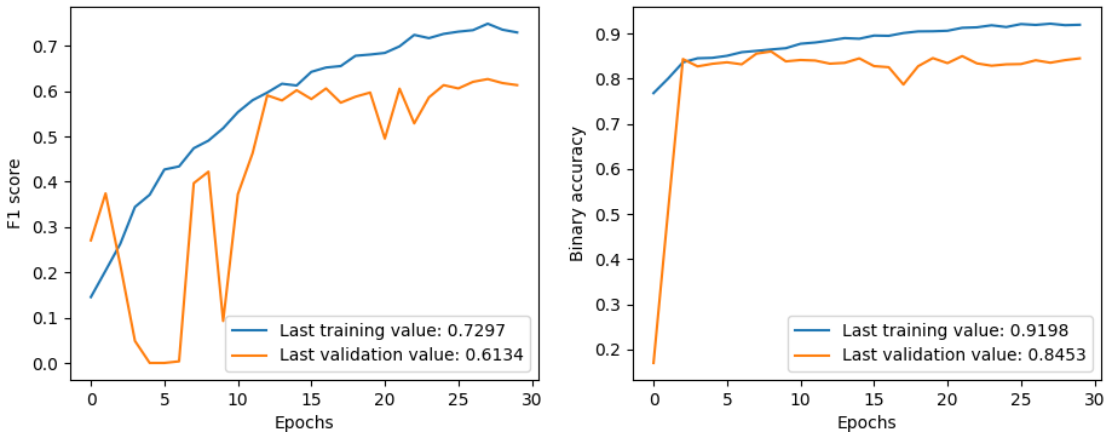



Figure C.2: Learning curves of experiment two from Table 6.3 with a dataset containing 16.09% of positive samples.

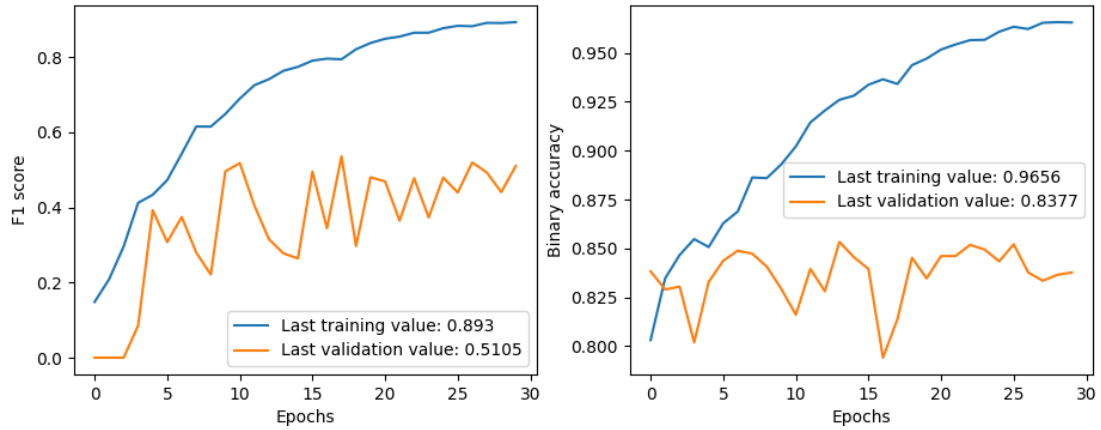
## C.2 Impact of learning rate

Figure C.3 depicts the learning curves of experiments three and five of Table 6.3. Those two differ regarding their learning rate, and their curves justify the choice of this one. Indeed, even though an initial learning rate of 0.1 seems better in Table 6.3, the choice was based on the curves. Therefore, as Figure C.3a shows better metrics values than Figure C.3b, the kept learning rate value was 0.01.

Several other experiments concerning the learning rate were made at multiple stages of the tuning process, notably after finding the best combination of hyperparameters. Table C.1 contains the details about some of them, showing that the chosen learning rate is indeed giving the best results.




(a) Initial learning rate = 0.01.




(b) Initial learning rate = 0.1.

Figure C.3: Learning curves of experiments three and five from Table 6.3.

| | Dataset | Initial LR | Batch size | Optimizer | Data augmentation | Dropout | Sampling | Class weights | Bias | AUC | F1-score | Accuracy |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | **15.38% (19946 img)** | 0.01 | 256 | Adam | ✓ | ✓ | ✓ | ✓ | ✓ | 0.92676 | 0.81286 | 0.85934 |
| 12 | 15.38% (19946 img) | **0.001** | 256 | Adam | ✓ | ✓ | ✓ | ✓ | ✓ | 0.81086 | 0.63658 | 0.68132 |
| 13 | 15.38% (19946 img) | **0.1** | 256 | Adam | ✓ | ✓ | ✓ | ✓ | ✓ | 0.92552 | 0.76696 | 0.82637 |

Table C.1: Table of additional Case2 experiments. The gray line corresponds to the best one. The bold values mark the changes between experiments.

**UNIVERSITÉ CATHOLIQUE DE LOUVAIN**
**École polytechnique de Louvain**
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**