Deep Learning Techniques for Content-based Medical Image Retrieval

Cézanne Alves Mendes Motta

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

Cézanne Alves Mendes Motta

Deep Learning Techniques for Content-based Medical Image Retrieval

Dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Master in Science. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Profa. Dra. Agma Juci Machado Traina

USP – São Carlos
May 2022

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

```
        Motta, Cézanne Alves Mendes
M921d      Deep Learning Techniques for Content-based
        Medical Image Retrieval / Cézanne Alves Mendes
        Motta; orientadora Agma Juci Machado Traina. -- São
        Carlos, 2022.
           95 p.


           Dissertação (Mestrado - Programa de Pós-Graduação
        em Ciências de Computação e Matemática
        Computacional) -- Instituto de Ciências Matemáticas
        e de Computação, Universidade de São Paulo, 2022.


           1. Deep Learning. 2. Content-based Image
        Retrieval. 3. Medical images. 4. Variational
        Autoencoder. I. Traina, Agma Juci Machado, orient.
        II. Título.
```

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Cézanne Alves Mendes Motta

Técnicas de aprendizado profundo para busca de imagens médicas por conteúdo

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Profa. Dra. Agma Juci Machado Traina

USP – São Carlos
Maio de 2022

ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001

RESUMO

MOTTA, C. A. M. **Técnicas de aprendizado profundo para busca de imagens médicas por conteúdo**. 2022. 95 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

No contexto de Diagnóstico Assistido por Computador (CAD), às vezes não é suficiente que o sistema produza predições corretas. Quando os médicos estão seguros a respeito do diagnóstico de um caso em particular, eles podem aceitar ou desprezar a predição do sistema de acordo com suas próprias conclusões. Mas em casos em que os médicos estejam inseguros, eles podem não confiar na predição do sistema sem que haja alguma explicação para ela. No domínio médico, onde os usuários são ética e legalmente responsáveis por suas decisões, o sistema deve ser capaz de articular de alguma forma as razões de suas decisões. Uma estratégia que tem sido sugerida para prover esse suporte à decisão é a de recuperar imagens similares que já foram diagnosticadas. Dessa forma, os médicos podem, então, comparar os casos retornados ao caso em consideração e decidir se os diagnósticos desses se aplicam. Tradicionalmente, Recuperação de Imagens Médicas por Conteúdo (CBMIR) tem sido feita com representações projetadas manualmente. Apesar de demonstrar melhorias significativas em muitas outras tarefas de análise de imagens médicas, *Deep Learning* (DL) não é frequentemente utilizado em CBMIR. A maioria das abordagens atuais para integrar DL em CBMIR utiliza representações obtidas de modelos treinados para classificar imagens. Esses modelos tendem a aprender representações que capturam características que são correlacionadas com as classes e ignoram as características que não o são. Apesar de serem úteis parar classificar imagens, essas representações ignoram variações intra-classe que podem ser relevantes para encontrar imagens visualmente semelhantes. O ideal seria retornar casos com a maior similaridade visual possível, para que médicos possam ter mais confiança nas suas decisões, e não somente casos que pertençam à mesma classe. *Autoencoders*, por outro lado, são modelos de DL que visam aprender representações que descrevam fatores de variação intrínsecos do conjunto de dados. Este trabalho visa investigar e discutir o uso de DL para busca de imagens médicas, apresentando a fundamentação teórica e análise crítica das abordagens atuais encontradas na literatura. Também é apresentada uma abordagem para CBMIR baseada em *Variational Autoencoders* e mostrado que essa abordagem pode produzir resultados superiores àquelas baseadas puramente em classificação, e pode inclusive ser utilizada em conjunto com estas.

Palavras-chave: Aprendizado profundo, Busca por imagem baseada em conteúdo, Imagens médicas.

ABSTRACT

MOTTA, C. A. M. **Deep Learning Techniques for Content-based Medical Image Retrieval** . 2022. 95 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2022.

In the context of Computer-Aided Diagnosis, it is often not enough for the system to produce correct predictions. When physicians are certain about the diagnosis of a given case, they can accept or disregard the system's prediction according to their own conclusions. However, in cases where they are uncertain, the physicians may not trust the system prediction without an explanation for it. In the medical domain, where the users are ethically and legally responsible for their decisions, the system should be able to articulate the reasons for its prediction in some way. One strategy that has been suggested to provide this support for decision is to retrieve images from similar cases that were already diagnosed. The physicians can then compare the retrieved cases to the one under consideration and decide if such diagnosis apply. Traditionally, Content-Based Medical Image Retrieval (CBMIR) has been done with hand-crafted features. Despite showing significant improvements in many other medical images analysis tasks, Deep Learning is not frequently used in CBMIR. Most current approaches to integrate Deep Learning into CBMIR use features obtained from models trained to classify images. These models tend to learn features that are correlated with the classes and ignore the ones that are not. Despite being useful to categorize images, the features learned from such models ignore intra-class variations that may be relevant to finding visually similar images. The ideal would be to retrieve the most visually similar case possible, not just one that belongs to the same class, so that the physicians can have more confidence in their decision. Autoencoders, on the other hand, are Deep Learning models that aim to learn features that describe the intrinsic factors of variations of a dataset. In this work, we investigate and discuss the use of Deep Learning for medical image retrieval, presenting the theoretical foundation and a critical analysis of current approaches found in literature. We also propose an approach for CBMIR based on Variational Autoencoders and show that this approach can yield better results than the ones based solely on classification and can even be used in combination to improve the results of the latter.

Keywords: Deep Learning, CBIR, Content-based Image Retrieval, Medical images, Variational Autoencoder.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

* *

CHAPTER

1

INTRODUCTION

Computer-Aided Diagnosis (CAD) emerged as an alternative for early attempts of automated diagnosis. In this new approach, it is assumed that the output of the computer should be utilized by radiologists, not replace them. The computer output is used as a second opinion, and the physicians make the nal decision (DOI, 2007).

To produce a useful output for such process, one of the approaches that have been suggested is the use of Content-Based Image Retrieval (CBIR) systems to retrieve images similar to the case been analyzed. The idea is that, by searching for previous cases whose images are similar to that of the case at hand, the physician can use the retrieved cases for doing a comparative diagnosis, adding more information and con dence to the analysis and evaluation process, which supports the specialist in the nal diagnosis (TRAINA et al., 2017)

This approach to Computer-Aided Diagnosis has some bene ts to the approach of pure classi cation, even when the physician is trying to con rm or rule out a certain class. While, with classi cation, the computer simply says whether it thinks the image belongs to certain classes of interest, with Content-Based Medical Image Retrieval (CBMIR), the physician can look at the similarities between the retrieved images and the queried one to get a sense of why the case in hand should belong or not to a class or another. Instead of automated predictions of classes, the retrieved previous cases may have complete diagnoses and reports made by other certi ed physicians. And by analyzing the evolution of past similar cases, the physician can better estimate the prognosis of the case at hand and the outcomes of possible courses of action.

The retrieval is achieved by means of a function $f(x) = z$, called a descriptor or feature extractor, that maps the images x to a vector representation z, called a feature vector, that can be used with a distance function to compare the images from the domain. If the descriptor used is appropriate for the task and image domain in question, the distance between images that are semantically more similar will tend to be smaller than between those that are less similar, and a query can be performed by computing the feature vector $z_q = f(x_q)$ of a query image x_q and, either ranking the pre-computed feature vectors of the images of the database in increasing distance to z_q, or performing a neighborhood query with an efficient indexing method.

Traditionally, CBIR on the medical domain has been done by comparing automated descriptions of the contents of the images obtained by carefully hand-designed algorithms. Although popular and effective in many applications, these hand-crafted features have some shortcomings. Since these algorithms are made with standardized fixed rules, they need to be based on expert knowledge, which can be expensive and time-consuming, have a narrow application domain and don't generalize well to large scale data where there may be outliers not covered by these rules (LI et al., 2018).

Deep Learning methods, on the other hand, are able to learn algorithms that produce high-level representations from the data itself (LECUN; BENGIO; HINTON, 2015). Today, the most successful type of models for images analysis and the top performers in most medical image analysis competitions are based on Deep Learning (LITJENS et al., 2017; HU et al., 2018). Following this success in medical image analysis, many techniques were proposed to obtain descriptors for CBMIR from Deep Neural Networks.

However, most of these techniques rely on Convolutional Neural Networks (CNNs) trained to perform image classification (a task so commonly associated with such network architecture that is usually implied by its use) or else trained in more elaborated ways that also rely on hit rate over a certain class of interest. Models trained in such way suffer of two important disadvantages. The first disadvantage is that those models require a large amount of accurate and structured annotations for class supervision. This is a challenge for the medical domain because these annotations can only be provided by certified physicians specialized at the domain in question, which is expensive. Often hospitals have databases with large amounts of image data of previous patients, but there is no structured annotation available. Such databases could be used for CBMIR once we have the descriptors, but to train descriptors using the classification approach, they first would have to be annotated by a committee in an arduous and expensive process. Because of this, unsupervised methods for generating descriptors with performance close to that of supervised models would be highly valuable for CBMIR. The second disadvantage is that models trained in classification tend to miss out on visual information that, although not relevant for image classification, may be relevant for CBMIR.

Motivated by this context, in this work, we investigate and discuss the use of Deep Learning for medical image retrieval, presenting the theoretical foundation and a critical analysis of current approaches found in literature. We also propose an alternative approach for using Deep Learning for CBMIR to mitigate said disadvantages, and show that this approach can yield better results than the ones based solely on classification and can even be used in combination to improve the results of the classification approach.

1.1 Scope

This work has its scope of study de ned in the intersection between

- A technique: Deep Learning
- A task: Content-Based Image Retrieval
- A Domain: Medical images

Within that scope, this work is driven by the following research question:

“How to use Deep Learning for Content-Based Medical Image Retrieval in a way that is useful for Computed Assisted Diagnosis?”

In particular, during the study of theoretical background, the class of Deep Learning architectures known as Variational Autoencoders (discussed in section 3.5.3.2) was identi ed as potentially applicable for this purpose. However, despite the theoretical applicability, during the literature review no works where found investigating the viability of Variational Autoencoders for this application. Hence, the proposed research aims at investigating the following hypothesis

“Can features from Variational Autoencoders be used to improve upon current medical image retrieval approaches?”

1.2 Outline

The following chapters of this document are organized in the following manner:

- Chapter 2 introduces concepts of CBIR, the metrics used to evaluate such systems, and examples of hand-crafted features traditionally used.
- Chapter 3 introduces elements of Deep Learning techniques and commonly used architectures. It also includes, in section 3.5, a theoretical discussion on how Autoencoders differ from classi ers on learning representations from data.
- Chapter 4 brings the main discussion to support the proposal. It presents motivations for using image retrieval to assist diagnosis and for pursuing the use of Deep Learning techniques for this task. It also presents the current approaches found in literature review and why the use of Variational Autoencoders may improve upon them.
- Chapter 5 Describes the proposed approach for learning descriptors for CBMIR, experiment methodology and results comparing it to the approach of classi cation.

CHAPTER

2

CBIR

2.1 Introduction

A Content-Based Image Retrieval (CBIR) system retrieves images from a database by analyzing their content instead of relying exclusively on labels, textual descriptions and metadata. So, for example, a system that classify images into disjoint or overlapping categories and satis es queries by categories ful l this de nition since the categories were not pre-existing metadata. For the system to label these images, it has to analyze their contents to some extent, and so the system is sensitive to content information and does not rely totally on pre-existing textual or structured metadata do retrieve the desired images.

In this work, however, we are interested in the mechanisms of query by example(QBE) andsimilarity search, and the motivation for that is discussed in chapter 4. Nothing prevents this mechanism to be coupled with ltering by class, or the inclusion of structured information in the query or in the database, but QBE and similarity search are crucial to better satisfy the needs of the intended application.

2.1.1 Query by example and similarity search

In QBE CBIR systems, the user provides an example image which will be used to inform the system of the types of images he or she wants returned. The system could infer the class of the image and return images of the same class and this would constitute QBE, but in similarity search the system would prioritize the most similar images. It could rank the entire dataset from the most to the least similar, or it could return the k most similar ones.

For the system to measure similarity, the images must be transformed in, or represented by something quantitatively comparable. This is usually achieved by generating a vector that describes the image in many aspects or dimensions, a vector in which each dimension represents somefeature of the image – hopefully relevant to the target domain – and the functions designed to generate these *feature vectors* are called *descriptors*.

To rank the images, a *similarity function* – usually a decreasing function of a distance function, such as the Euclidean distance – is used to compare the feature vectors. A feature vector with d dimensions can be seen as a point in $\mathbb{R}^d$ space, and so the similarity is usually measured by a distance function in this vector space, such as the Euclidean distance. In this text, a pair of descriptor and similarity function will be referred as *comparator*, the same name will be used to functions that output a distance or similarity level to raw images without using descriptors.

The database to be queried is pre-processed of ine where the feature vectors from each image is computed and indexed in a data structure with permits an ef cient execution of a nearest neighbor query. That way, when the user makes a query, the system extracts a feature vector from the provided query image and retrieve the images whose feature vectors are nearest to the query point (SMEULDERS *et al.*, 2000).

There are many indexing data structures, query types and similarity functions used in CBIR literature. However, since this work aims to apply Deep Learning to the task of QBE and similarity search, we will focus the discussion on the components of conventional CBIR systems that differ signi cantly to the proposed method using Deep Learning, that is the feature extractor. And so, we will assume the use of the Euclidean distance to measure similarity and simplify the query process to sorting the database images with increasing distance from the query point.

Figure 1 – Illustration of similarity search in a traditional CBIR system

Source: Elaborated by the author.

2.2 CBIR evaluation

To objectively evaluate the suitability of an image descriptor to CBIR in a given application domain, an image dataset with ground-truth information is required. The ideal ground-truth information to evaluate an image descriptor would be an all-to-all distance matrix (or similarity matrix) with the ideal distances for each pair of images in a dataset. That way we could measure how close to the true distances (or similarities) the ones provided by the descriptor are, or how close they are to be proportional, or how similar are the rankings produced by them.

Dataset with annotations like this are hard to nd and even to produce: For many applications, there is no objective standard of similarity, not even ordering by relative similarity. Even asking domain specialists to rank a set of images in order of similarity to an example image can yield different rankings. For that reason, the performance[1] of CBIR systems is usually evaluated with datasets annotated with class labels, and the ability of the descriptor to accurately measure relative similarity of images is estimated by its measured ability to discriminate classes of images.

Two commonly used measures are the precision x recall curve (PR-curve) and the Mean Average Precision. To use those measures in the context of QBE with class labeled datasets, we consider relevant all images with the same class C_q as the query image.

2.2.1 Precision and Recall

Precision and recall are measures of relevance of a set of retrieved elements for a single query. Precision measures how much of the set retrieved by the query is relevant and Recall measures how much of relevant elements in the whole dataset were retrieve by the query.

To formalize, let us de ne 4 categories:

- True positives (Tp) as the set of relevant images retrieved
- False positives (Fp) as the set of irrelevant images retrieved
- True negatives (Tn) as the set of irrelevant images not retrieved
- False negatives (Fn) as the set of relevant images not retrieved

From that: we de ne precision and recall as:

$$\text{Precision} = \frac{Tp}{Tp + Fp}$$

$$\text{Recall} = \frac{Tp}{Tp + Fn}$$

Note that the order in which the elements are retrieved doesn't affect those measures. Moreover, if the query ranks and returns the entire dataset, recall will be always equal to 1 and precision to the percentage of the class C_q in the dataset, nothing that depends on the descriptor.

1 In the present document, performance refers to measures of relevance of retrieval such as precision and recall. Computing performance is referred to as ef ciency. Nomenclature was chosen that way to be consistent with reference literature.

Figure 2 – Illustration of precision and recall measures

Source: commons.wikimedia.org

For that reason, precision and recall are used in similarity search to evaluate a K-nearest neighbors query. When querying for a xed k, we denote Precision at k as P@k and Recall at k as R@k.

2.2.2 PR-curve and Average precision

To evaluate a ranking of a single query, we can use precision and recall as if returning only the rst k elements of the ranking for k varying from 1 to the number of elements in the entire dataset. Then we draw a parametric curve of P@K and R@k in function of k.

Now this curve may be used to evaluate the ranking because Precision will only reach 1 if the rst element is relevant and will remain closer for longer if the relevant elements are ranked higher.

The values of precision after recall reaches 1 are not meaningful because there are no relevant elements left to be returned. Therefore, its useful to see precision in function of recall, which is possible since recall is monotonically non-decreasing in function of k.

By plotting precision in function of recall, we get a better visual description of the relevance of the ranking: the precision curve is normalized according to the number of relevant elements, and a ranking with all relevant elements before the irrelevant ones is guaranteed to have an area of 1 under the curve.

The area under the curve, on the other hand, gives a single number metric for the relevance of the ranking and is known as Average Precision (AP) and given by:

$$AP = \int_0^1 p(r)\,dr$$

Which is equivalent to

$$AP = \sum_{k=1}^{n} P(k)\,\Delta r(k)$$

$$\text{where } \Delta r = R(k) \quad R(k \quad 1)$$

Or

$$AP = \frac{\sum_{k=1}^{n} P(k) \quad rel(k)}{\sum_{k=1}^{n} rel(k)}$$

where $rel(k) = 1$ if the k-th item is relevant and 0 otherwise.

Now, to evaluate the descriptor both in KNN queries or in ranking, we can compute the mean of the appropriate measures across all the queries obtained as taking each image of the dataset as the query image. The area under the mean precision recall curve for the whole dataset is the measure known as mean Average Precision (mAP)

$$mAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}$$

2.3 Global Descriptors

A global feature is a feature that can be extracted from the whole image without requiring (although sometimes used with) pre-processing steps or segmentation for it to be applicable.

2.3.1 Intensity Histogram

The gray-level histogram of an image with L intensity levels is a discrete function $h(r_k) = n_k$, where r_k is the k-th intensity value and n_k is the number of pixels with intensity r_k.

The histogram is usually represented in the computer as an array and can be used directly as a L-dimensional feature vector and be compared to other histograms of same dimension by a distance function. In natural images (or medical images in a common le format), usually L=256.

Sometimes vectors with less dimensions are desired, especially if it will be concatenated with other feature vectors (or if other color channels are used). This can be achieved by reducing the number of intensity levels on the image.

Another option to reduce the size of the feature vector is to compute statistics from the histogram to describe the probability distribution. By dividing the entries of the histogram array, we get the normalized histogram $p(r_k) = \frac{h(r_k)}{m \quad n}$ which gives the probability that a randomly selected pixel will have intensity r_k. The normalized histogram is, therefore, a probability distribution and can be summarized by common statistical moments. A small feature vector can then be made with a few statistics of the histogram. However, such feature vector would carry less information about intensity than the full histogram.

In the case of color images, one option is to generate one histogram for each RGB channel and concatenate them into one feature vector. Another option is to convert the images to HSV color-space where we can compute histograms for hue, saturation and value (intensity), which may be semantically more relevant to the application domain.

Despite describing intensity levels or color distribution, the histogram does not take pixel location into account, it can only describe the frequency of a color in the image but not where the color is in the frame. That is, histograms do not keep information about the spatial distribution of the image.

2.3.2 Haralick Features

The Haralick features (HARALICK; SHANMUGAM; DINSTEIN, 1973) are statistics computed over a gray-level co-occurrence matrix. They capture texture information by taking into account not only the intensity levels of the images, but also their relative position.

Loosely speaking a gray-level co-occurrence matrix Q counts how many times each pair of intensities occur in a given arrangement in an image. To formalize that, let us de ne Displacement $D(k,l) : \mathbb{N}^2 \rightarrow \mathbb{Z}^2$ as a displacement in the domain of matrix indices, such as the displacement $D(k,l) = (k, l+1)$ of “one pixel to the right”, for example. We say that two pixels are D-neighbors if one of them is at displacement D of the other, and represent that by the Boolean operator $\underset{D}{\leftrightarrow}$:

$$(k,l) \underset{D}{\leftrightarrow} (m,n) \Leftrightarrow (D(k,l) = D(m,n) \text{ OR } D(m,n) = (k,l))$$

With the notion of D-neighborhood, we can formalize the co-occurrence matrix Q of a displacement D as the matrix that counts the number of times that intensities i and j appeared in D-neighboring pixels:

$$Q_{i,j}(D) = \left|\{((k,l),(m,n)) : (k,l) \underset{D}{\leftrightarrow} (m,n) \text{ AND } I_{kl} = i \text{ AND } I_{mn} = j\}\right|$$

Note that i and j are not pixel indices, but intensity levels of the pixels, they are indices only on the matrix $Q(D)$. Also, Q is de ned in terms of the D-neighborhood instead of on the displacement so that $Q_{i,j}(D) = Q_{j,i}(D)$, therefore Q is always symmetrical and invariant to the direction of the displacement.

Returning to the example of the displacement $D(k,l) = (k, l+1)$, Figure 3 shows an example image with its co-occurrence matrix Q(D).

Figure 3 – Example of co-occurrence matrix.

Description: Left: a 4x4 image with 4 levels of intensities. Right: a co-occurrence matrix of the intensities of the image on the left

Now, by dividing Q by the total number of occurrences, we get a matrix P whose elements $P_{i,j} = p(i,j)$ denote the probability that if we take a random pair of D-neighboring pixels in the image they will have intensities i and j. From that joint probability distribution, many statistics can be computed to summarize the texture. The features proposed by Haralick consist of 14 statistics used to summarize this probability distribution. For simplicity, Table 1 contains the most used ones according to Felipe, Traina and Traina (2003). Each statistic produces one scalar feature, and so, assuming all haralick features being used, the process will produce a vector of 14 dimensions for displacement D.

Table 1 – Commonly used haralick features

Descriptor	Equation	Meaning
Variance	$\sum_i \sum_j (i-j)^2 P(i,j)$	Level of contrast of the image
Entropy	$\sum_i \sum_j P(i,j) \log P(i,j)$	Suavity of the image
Energy	$\sum_i \sum_j P^2(i,j)$	Uniformity of the image
Homogeneity	$\sum_i \sum_j \frac{P(i,j)}{1+\lvert i-j \rvert}$	Homogeneity of pixels distribution
3rd Order Moment	$\sum_i \sum_j (i-j)^3 P(i,j)$	Level of distortion
Inverse Variance	$\sum_i \sum_j \frac{P(i,j)}{(i-j)^2}$	Inverse level of contrast

Source: Haralick, Shanmugam and Dinstein (1973).

The original work proposed computing features for the co-occurrence matrices for the displacements of d pixels (i.e. L_∞ norm) in the directions of 0°, 45°, 90° and 135°. Since Q is invariant to the orientation of the displacement, the features of the opposing angles would be identical.

That way, we could generate one such feature vector for each displacement of 1 pixel in L_∞ norm in the directions of 0°, 45°, 90° and 135° and concatenate them into one feature vector if discriminating the orientations of textures is desired or averaging those in one feature vector if rotation invariance is desired. The parameter d of the displacement influences which size of texture patterns will be captured by the features.

The haralick features tend to capture texture information because, while intensity histograms only encode the frequency of intensity levels in an image, the co-occurrence matrix and its statistics encode the frequency to which intensity levels appear together with a given relative position. When a texture pattern repeats on the image, its intensity levels occur again at the same relative position to each other.

One limitation of using only haralick features is that it will not consider the location of the textures neither the primitive shapes composing them (SONKA; HLAVAC; BOYLE, 2014), when there are textures composed of larger primitive shapes in the image.

2.3.3 Zernike Features

The Zernike features (TEAGUE, 1980) are shape features that are not based on the contour of a segmentation, they are moments taken from the image function to represent its shape and give an overall description of how intensity is distributed across the image. These moments have a property intrinsically invariant to rotation and can be used to extract features that are insensitive to the orientation, position and scale of an object in an image.

The Zernike moments are the complex coefficients associated with each Zernike polynomial (ZERNIKE, 1934) in a transformation having those polynomials basis vectors. The basis formed by these polynomials is orthogonal and the procedure is similar to the Fourier transform, only with a different basis.

The Zernike polynomials (illustrated in Figure 4) are a series of functions defined only within the $\mathbb{R}^2$ unitary circle. Each polynomial V_{nl} is identified by its degree n and angular dependence l and defined by

$$V_{nl}(x,y) = \sum_{m=0}^{\frac{n-|l|}{2}} (-1)^m \frac{(n-m)!}{m!\left(\frac{n-2m+|l|}{2}\right)!\left(\frac{n-2m-|l|}{2}\right)!}\left(x^2+y^2\right)^{\frac{n}{2}-m} e^{il\theta}$$

where $x^2+y^2 \leq 1$, $\theta = \tan^{-1}\left(\frac{y}{x}\right)$, $i = \sqrt{-1}$, $|l| \leq n$, and $n-|l|$ is even.

Suppose now that we want to project a gray-scale image function F[i,j] into the basis obtained by a set of Zernike polynomials. These polynomials are known to be orthogonal, and this property allows to compute the coefficient for each component in the projection independently.

First, the image pixels must be mapped by ϕ to points in the real unit disc since Zernike polynomials are defined only in that region. Let $f(x,y) = F(\phi^{-1}(i,j)) = F[i,j]$ be a function defined only on the points (x,y) where the pixels of the image landed. The component of the Zernike transform of $f(x,y)$ associated with the basis vector V_{nl} would be:

$$Z_{nl} = \frac{n+1}{\pi}\sum_{x}\sum_{y} V^{*}_{nl}(x,y)\, f(x,y)$$

where V^{*}_{nl} is the complex conjugate of V_{nl}, and the summations iterate over the points (x,y) of the unit disc where the pixel of the image landed.

Figure 4 – Illustration of the rst 21 Zernike polynomials

Source: commons.wikimedia.org

These coef cients are the Zernike moments. Their values give a notion of how the mass (or intensity) is distributed across the unit disc. The lower degree coef cients account for higher mass regions, the higher degree coef cients account for the ner details as can be seen in Figure 4.

2.3.3.1 Comparison with contour-based shape features

When compared to contour-based shape features, the Zernike moments have some advantages since they do not rely on describing the contours of a segmentation. Suppose we have a single contiguous object segmented from an image, but the shapes of the internal parts of the object are semantically relevant to the application. With a contour-based shape feature, each of those parts would have to be independently segmented, identi ed and have its contour features extracted.

Besides representing the shape of an object, the Zernike features can represent the shape of its internal parts and their overall intensities as well. Moreover, they can represent objects composed of disjoint regions, or even a scene with multiple objects if the arrangement of the objects is relevant to the application. All those scenarios could be represented by the moments of a single Zernike transform without the need to independently segment every part whose shape is relevant. It is important to note that, despite these advantages, the Zernike moments will be affected by every intensity value inside the unit disc, and only by them, therefore any background irrelevant to the application should be removed, and all relevant regions must be mapped to the unit disk.

To illustrate the information that the Zernike moments encode, Figure 5 shows a reconstruction of a brain MRI image with the rst 121 complex coef cients of both a Zernike transform and a Fourier transform.

Figure 5 – Reconstruction of an image from Zernike

Source: Elaborated by the author.

Description: Reconstruction of an image from coef cients of a Zernike transform compared to a Fourier transform. Left: Original image. Middle: reconstruction from the Zernike coef cients. Right: reconstruction from the Fourier coef cients

2.3.3.2 Invariances

The coef cients obtained by the Zernike transform are complex and their magnitudes are invariant to rotation of the original image. Therefore, rotation invariant features can be obtained by taking the magnitude of the complex coef cients obtained by the Zernike transform. Also, if only the magnitudes of the coef cients were to be used as features in the intended application, then the transform can be done with a basis of only polynomials V_{nl} with $l \geq 0$. As can be seen in Figure 4, the polynomials V_{nl} and $V_{n,-l}$ are identical except for rotation.

To achieve translation invariance, all images should be centered in the unitdisc in the same way, so that images that are identical except for their translation end up similar after centering. Usually the center of mass (considering pixel intensity as mass) is used as the center of the unit circle, as it does not depend on the rotation of the image.

Scale invariance may be achieved in a similar manner. If the objects of the intended application have well de ned contours the images can be scaled by maximum diameter or else – if they are being centered as well – can be scaled by maximum radius from the center. If the objects do not have well-de ned contours, one possibility is to use the standard deviation of the weighted distances to the center of mass, named here as *standard weighted radius* s_{tr} and de ned as:

$$s_{tr} = \sqrt{\sum_{x,y} f(x,y)\,\left((x-x_0)^2+(y-y_0)^2\right) \Big/ \sum_{x,y} f(x,y)}$$

where $f(x,y)$ is the intesity at pixel (x,y) and (x_0,y_0) is the center of mass.

2.4 Chapter Considerations

This chapter brie y summarized the main concepts of CBIR needed to follow the research presented in this dissertation, such as the general QBE and similarity search frameworks and evaluation methods. There are other aspects of the eld that were not discussed here, such as other approaches for retrieval besides QBE, ef cient indexing methods and many other strategies for generating feature vectors.

Many of those strategies consist of tailoring image descriptors speci c for each domain by pipelining several different local algorithms to describe parts of interest of the image. For example, one could describe a region of interest such as a medical nding by its pixel area, or the structure of its shape by a "skeleton" that spans the entire shape reaching to its borders, or the roughness of its border by describing the frequency of its oscillations. But in order to do this, one would rst need to detect such nding on the image and de ne its border, which is not always simple when the object of interest has cavities or tangled rami cations, such as vascular structures. And then, we would still have the problem of what to do when no such nding is detected, or how to fuse feature vectors when many disjoint regions are found, and how to fuse that to feature vectors representing other aspects of the image, such as texture. And all of that would have to be done in a way that makes sense semantically for the domain and the real-world objects that we are trying to characterize, so that semantically similar images would have similar feature vectors.

Because of these many details, those pipelines of local descriptors usually are designed and tuned for speci c domains and objects, and usually will not work well in other contexts, and, therefore, are not discussed here in the context of a general approach for medical images, but the interested reader may nd more information about local descriptors in Gonzalez and Woods (2008), Prince (2012). We instead presented only global descriptors for color texture and shape mainly as examples of classical techniques that can be used directly on the image and work generally across many image domains. While this generality brings the advantage that the descriptor will work with images from any domain, it has the disadvantage of possibly loosing image characteristics that are important for the given task and domain, and letting other factors that are not relevant interfere on the resulting feature vectors.

Contrary to the classical methods, Deep Learning techniques allow the automated learning of speci c domain representations directly from a dataset. That way we have the advantage of a machine learning model that works generally well across many domains and tasks, while the output of the learning process is speci c to the domain in which it was trained. Currently, this approach is employed in state-of-the-art solutions, surpassing classical descriptors, both global and local, in most computer vision tasks in medical images (LITJENS et al., 2017). In the next chapters we present the fundamentals of Deep Learning and how they can be used for learning image descriptors, and how Deep Learning has been used in the literature for medical image retrieval.

CHAPTER

3

DEEP LEARNING

Machine learningalgorithms are algorithms that can learn from data (GOODFELLOW; BENGIO; COURVILLE, 2016).Deep Learningis a special case of machine learning, where the algorithm learns multiple levels of representation, obtained by composing simple but non-linear modules, each of which transforms the representation at one level (LECUN; BENGIO; HINTON, 2015). This chapter brings de nitions and illustrations about the Deep Learning architectures that concern the discussions in chapters 4 and 5.

The concepts presented in this chapter are already consolidated and can be found on both Goodfellow, Bengio and Courville (2016) and Aggarwal (2018) – the books used as the main references for this chapter. The contents of this chapter are the following:

- Section 3.1 presents the machine learning concepts relevant to the Deep Learning architectures discussed in this chapter.
- Section 3.2 introduces Arti cial Neural Networks - the architectures used to model tasks in Deep Learning; presents the fully connected architectures, the classes of function that are capable of representing, and the task they can be used to model.
- Section 3.3 presents the gradient descent, the optimization procedure that allows the neural networks to learn, and the objective functions used in conjunction with the networks discussed in this chapter.
- Section 3.4 introduces Convolutional Neural Networks, the general architecture commonly used for tasks involving images; the operations typical of this architecture, and the effects of these operations on neural networks when processing images.
- Section 3.5 discusses issues regarding representation learning and the autoencoder architecture, used for learning the intrinsic features that describe a dataset.

3.1 Machine learning concepts

Many machine learning algorithms, and specifically the ones that will be considered in this chapter, consist of choosing a function h, between a set of admissible functions – called the hypothesis space – for executing a task. When the algorithm tries to learn a classification task from an example dataset, it is looking between its set of admissible functions for the function that best classify the example dataset in hope that the same function will work well to classify similar datasets.

A machine learning algorithm is typically composed of three parts: A model, an objective function, and an optimization procedure. The model usually consists of a parameterized family of functions $f(x; w)$. Each different value of the parameter w yields a different function. The performance of a candidate function is measured by a chosen criterion called *objective function* or *loss function* L which evaluates f over an entire dataset. And the best function of the family (or the best the algorithm can find) is obtained by optimizing L over w (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.1.1 Tasks and supervision

Machine learning algorithms are broadly categorized in supervised or unsupervised. However, this categorization has more to do with the information present in the dataset the algorithm is allowed to learn from than with the task it will perform (GOODFELLOW; BENGIO; COURVILLE, 2016).

In supervised learning the algorithm is allowed to experience a dataset of examples $(y_i; x_i)$ of inputs (x) of the task it will perform as well as the correct target output values (y) for that task. These target outputs are called *labels*. If it is desired to assign each new input to one of a finite number of discrete categories the task is called *classification*. If the output consists of a continuous value, then the task is called *regression*. In unsupervised learning the algorithm receives a dataset without the target output data of the task it will perform. Examples are tasks such as *clustering* where it is desired to discover meaningful groups on the dataset, and *representation learning*, where we try to find the best representation of the data according to a given criterion (BISHOP, 2006).

Despite supervision having more to do with the information the algorithms use for learning the task, the tasks have become synonyms with the supervision they receive. For example, if a clustering algorithm learns from a dataset of examples $(y_i; x_i)$, where y_i is the correct cluster for x_i to be assign to, such algorithm would just be called a classification algorithm. Similarly, a representation learning algorithm which learns continuum representations for a domain after learning from examples containing the correct representation would be called a regression algorithm.

3.1.2 Under tting and Over tting

In the case of supervised learning the model learns from a dataset with the intent of performing the task elsewhere since the set from where it learned already has the correct answer for each example. In this case we refer to the process of learning as “*training*” and to the dataset where it learned to do the task as the *training set*.

If the algorithm doesn't succeed to nd a function that performs well on the training set according to L, we say that the algorithm *under tted*. This could be due an ill-posed criterion on the objective function, the optimization algorithm failing to optimize the parameters, or because the set of admissible functions doesn't have good enough functions (according to the de ned criterion) for the task.

In the case of supervised learning, it can happen that the learned function performs well on the training set, but not on another data from the same domain. In such situation we say that the model *over tted* to the training set.

The training set can be seen as a random sample of the *data generating process*, and other samples can be slightly different. Over tting happens when the algorithm nds a function that model the idiosyncrasies of the training set instead of the underlying properties of the data generating process. This normally occurs when the model has too much *capacity* for the size of the training set. Informally, the capacity of the model is its ability to t a wide variety of functions and is related with its hypothesis space.

The ideal situation would be to use a model $f(x; q)$ with small capacity, capable of representing only a small family of functions that still contains the desired ones for the task. If we know what functions we need to do the task, the smaller the set of functions containing them, the better.

However, we don't always know so much about the function we are trying to learn, and we need to increase the search space. In those situations, there is usually a trade-off: Models with lower capacity tend under t, and models with higher capacity tend to over t (GOODFELLOW; BENGIO; COURVILLE, 2016).

Since the algorithm can over t, to know whether the function learned by the algorithm *generalizes* well to other datasets, we must *test* it in a dataset the algorithm was not allowed to train on. The *test set* need to have the correct target outputs for each example, so we can measure the performance of the algorithm on it, but the algorithm cannot have access to the labels of the test set.

Sometimes we don't know exactly the ideal family of functions for executing the desired task and need to increase capacity by widening the variety of admissible functions. But if we know of ranges of combinations of parameters values or that tend to not yield good results, we can rule out these solutions by encoding restrictions on the algorithm, or by penalizing these parameters on L, therefore reducing the *effective capacity* of the model. Such approaches aimed at reducing over tting (but not under tting) are known as *regularization*.

3.1.3 Hyperparameters and Validation set

Usually, the learning algorithm can be customized through some settings that affect the hypothesis space, the architecture of the model itself or some regularization imposed on it. These settings are called *hyperparameters* and should not be learned by the algorithm itself.

The reason is because learning algorithms chooses parameters that allows it to perform better on the training set. If the algorithm was allowed to control its own capacity, it would always increase it. Higher capacity makes easier for the model to perform well on the training set, but also makes it more likely to over t and perform badly on unseen data.

For that reason, hyper parameters are tuned outside the algorithm and independently of the training set. This *tuning* could be completely manual and done by the programmer, or by an automated routine that search for values randomly or in a grid manner.

If the values of hyperparameters are tested in the same data the algorithm is training, the same problem occur: the ones providing higher capacity would always perform better. Therefore, a third dataset is needed to perform the tuning of the model: the *validation set*.

The tuning is usually done by training the model (at least partially) in the training set with a con guration of hyperparameters, and evaluating its performance in the validation set, and that is repeated for many hyperparameter con gurations.

Even though the training procedure doesn't use the validation set, the tuning procedure, nonetheless, is also adapting the model on the basis of the validation set. This can also bias the model to perform better on the validation making the performance on the validation overestimate the performance on unseen data. The ner the search for hyperparameters, the more biased the model can be toward the validation set.

For that reason, there is still a need for a test set, where the model will be tested in examples that never in uenced it, in order to estimate the real performance of the model. A conventional partition ratio for the data is 2:1:1 for training, validation and test sets respectively.

The names “validation set” and “test set” can be somewhat misleading. It seems like we are testing parameters in the validation set, and validating the model in the test set, but the names are stuck by convention.

3.1.4 Feature learning

There are many linear models where the output is a linear function $f = w^T x + b$ of the input x, or yet generalized linear models, where a linear function of the input is fed to a well-behaved monotone non-linear activation function g to predict parameters of probability distributions over different classes, resulting in a model $f = g(w^T x + b)$. But even in the latter cases, *decision boundaries*, the boundaries that separate classes in the domain, are still a linear function of the input.

One way to extend linear models to nonlinear functions of the input, is to apply the linear model to a version of the input transformed by a nonlinear transformation $\varphi(x)$ that computes features of x. The question, then, becomes how to choose a transformation $\varphi(x)$ where a linear model will perform well.

The dominant approach before the advent of Deep Learning was to hand-design φ (GOODFELLOW; BENGIO; COURVILLE, 2016). This is the case with the classical computer vision approach where there is a *feature extraction* step, done with a variety of hand-designed descriptors chosen according to each case (PRINCE, 2012; SONKA; HLAVAC; BOYLE, 2014). The strategy with Deep Learning is to learn φ. Deep neural networks use many linear computing units with non-linear activation functions to approximate a wide range of functions (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.2 Neural network architectures

As seen in section 3.1, machine learning algorithms can usually be separated in model, optimization criterion and optimization procedure. The models used in Deep Learning techniques are known as Arti cial Neural Networks. The objective function and optimization procedure are discussed in section 3.3. In this section, let us assume they are given and discuss the model: the architecture of neural networks and what types of functions they can represent.

3.2.1 One Neuron

An Arti cial Neural Network (sometimes simply called *Neural Network* – NN) is a representational model that consists of interconnected computational units loosely inspired in biological neurons. As such we refer to these computational units simply as *neurons*.

The simplest neural network consists of a single neuron connected directly to the inputs of the model and it is referred to as *perceptron* (ROSENBLATT, 1958). Each of those connections has a *strength* or weight w that can be any real number. The perceptron computes a *pre-activation* value z by performing a dot product between its input vector x and its weight vector w and adding an offset b, known as *bias* input. Equivalently, the bias input can be viewed as adding a dummy input x_0 with value 1 to the input vector and including an entry w_0 in the weigh vector for this input.

The bias input is of fundamental importance, because it acts as an intercept term and permits that z assumes any linear function of the input by varying the parameters w and b: $z = w^T x + b$. The result of this linear function is then fed to an activation function $g : \mathbb{R} \to \mathbb{R}$, which doesn't have any parameters. Therefore, the neuron can be represented by this simple

Figure 6 – Illustration of a perceptron

Source: MIT 6.S191: Introduction to Deep Learning http://introtodeeplearning.com/

equation:

$$f(x; w, b) = g(w^T x + b)$$

By changing the activation function $g(\ )$ we get different models, with different sets of admissible functions. For example, one can use the identity activation function $g(z) = z$ (which is equivalent to use none) to make the model be able to represent any linear function of x. Such model can be used to do linear regression over pairs of values (x, y) by predicting $y = f(x; w, b)$. Any linear activation function results in the same model since z already can represent any linear function of x and every composition of linear functions also results in a linear function. For such reason a neuron with the identity function, or any other linear function is called a linear neuron.

For binary classi cation, a threshold activation function such as the sign function can be used:

$$\mathrm{sgn}(z) := \begin{cases} 1 & \text{if } z < 0 \\ 1 & \text{if } z \ \ 0 \end{cases}$$

where we can consider output 1 as predicting that x belongs to one class and output 1 as predicting that it belongs to the other. The decision boundary then would be the hyperplane where $z = 0$. Note once more the importance of the bias term also in classi cation tasks: by changing the intercept term of $z = w^T x + b$, we also change the location of the decision boundary $z = 0$, allowing it to leave the origin.

If it is desired to model the probabilities that x belongs to each class, the logistic sigmoid function $s(\ )$ (often called just sigmoid function) can be used as an activation function to predict the probability that x belongs to one class. That is the model predicts that $p(y = 1 | x) = f(x; w, b) = s\ \ w^T x + b\ \ ;$ where $y \in f\ \ 1; 1g$ is the correct label class for x, and

$$s(z) = \frac{1}{1 + e^{\ z}}$$

By exclusion, the probability for the other class is then predicted to be $p(y = -1|x) = 1 - p(y = 1|x)$. Then, if we want to predict a class, we predict $\hat{y}$ to be the class with higher probabilities. By using $\sigma(\cdot)$ as activation function, the perceptron becomes equivalent to the classification model called logistic regression.

Note that despite $\sigma(\cdot)$ being a nonlinear function, the decision boundary of logistic regression (or a perceptron using $\sigma(\cdot)$ as activation function) is linear. That is because, despite $\sigma(\cdot)$ having continuous values, we still use an interpreting threshold $\sigma(w^T x + b) = 0.5$, which separates the regions in x where one class has higher probability than the other. Since $\sigma(\cdot)$ is a monotone function (which is the case for activation functions of generalized linear models (BISHOP, 2006; DOBSON; BARNETT; BARNETT, 2008)) the threshold will still correspond to a fixed threshold on $z = \sigma^{-1}(0.5) = 0$. This is illustrated in figure 7.

Since the bias term can be represented by adding a dummy input x_0 to the input vector and a corresponding weight w_0 to the weight vector, or by summing a bias term inside the neuron, the figures in the following sections will stop illustrating the bias input. However, it has to be kept in mind that, in the architectures presented in this section, every neuron has a bias term of its own.

Figure 7 – Illustration of a decision boundary produced by a perceptron with a sigmoid activation unit

Source: MIT 6.S191: Introduction to Deep Learning http://introtodeeplearning.com/

3.2.2 One Layer

To perform linear regression of vector valued functions, to predict independent variables with linear dependence on the input or to perform classification over more than 2 classes, more outputs are needed, so a layer of neurons is used, and the model can be represented by

$$\hat{y} = f(x; W; b) = g(z) = g(Wx + b)$$

where $g : \mathbb{R}^n \to \mathbb{R}^n$ is defined to operate elementwise over the components of its domain.

A special case occurs with classification, however: If it is desired to perform multi-label classification, where element x can belong or not to each class y_i independently, then the $\mathrm{sign}(\cdot)$

Figure 8 – Illustration of a neural network with two neurons in a single layer

Source: MIT 6.S191: Introduction to Deep Learning http://introtodeeplearning.com/

or $\sigma(\cdot)$ functions can still be used to perform the task as independent binary classi cation problems. However, if it is desired to predict a single class between all m classes, then such models can give invalid answers, such that an element belongs to more than one class or to none of them. In such case we can use a special activation function that does not operate element-wise, but takes all components of the input into account, the *softmax* function:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

The softmax function is a generalization of the sigmoid function for more than two classes. It models the probabilities that x belongs to each of the m classes and guarantees that the probabilities of all classes sum to 1.

3.2.3 Hidden layer

When using two or more layers of neurons, the last one is called the *output layer*, and the remaining layers and their neurons are said to be *hidden*. The default architecture is obtained by having all neurons in one layer connected to all neurons in the next one, and is commonly referred to as *multi-layer perceptron* (MLP).

The input of the network is sometimes referred to as the *input layer*, but it does not count to the total number of layers in the network because it does not perform any computation. This is just a notational convenience, as the neurons in the rst computational layer is connected to the input in the same way that any other computational layer is connected to the previous one.

To express networks with more than one layer, we use superscripts to denote the layer to which a variable belongs to and denote the outputs of the k-th layer as $h^{(k)}$. For example, a 2-layer MLP can be expressed as

$$\hat{y} = h^{(2)} = g\left(W^{(2)}h^{(1)} + b^{(2)}\right) = g\left(W^{(2)}g\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right)$$

Figure 9 – Illustration of a neural network with a hidden layer

Source: Adapted from MIT 6.S191: Introduction to Deep Learning http://introtodeeplearning.com/

3.2.3.1 Hidden units

In multi-layer neural networks, as in single-layer ones, the choice of activation function in the output units de ne the possible outputs of the network and are highly dependent of the task that is been modeled: usually linear activation for regression, sigmoid for binary or multi-label classi cation and softmax for multi-class classi cation. However, the activation functions in hidden layers serve a different purpose. They are used to add nonlinear transformations to the network and widen the representational power of the model.

Non-linear activation functions are essential in the hidden layers because an MLP with linear activation in the hidden layers, whatever its size, is equivalent to connecting all the inputs directly to the neurons in the output layer. That is because compositions of linear functions always result in linear functions, and the pre-activation value of the output layer is already capable of representing any linear function. For that reason, some non-linearity is always used in the hidden layers.

The most common ones are the already presented sigmoid function (s), the hyperbolic tangent function (tanh), and the recti er function. A neuron that uses the recti er function is called a ReLU (Recti ed Linear Unit) but the acronym is commonly used to refer to the function as well.

Typically, the ReLU is preferred because, at least for positive values, it doesn't saturate (decrease its derivatives to 0), allowing easier optimization with gradient descent (LECUN; BENGIO; HINTON, 2015). The mentioned functions are illustrated in gure 10 and de ned bellow.

$$s\,(z) = \frac{1}{1+e^{\ z}} \qquad \tanh(z) = \frac{e^{2z}\ \ 1}{e^{2z}+1} \qquad \mathrm{ReLU}(z) = \max f\,z; 0g$$

Figure 10 – Common activation functions

Source: Aggarwal (2018).

3.2.3.2 Universal approximation

Neural networks with a linear output layer and a single hidden layer of neurons with eligible nonlinear activation functions can approximate any continuous function on a closed and bounded subset of $\mathbb{R}^n$. The more neurons in the hidden layer, the closer the network can approximate the desired function

This property was initially proved (CYBENKO, 1989; HORNIK; STINCHCOMBE; WHITE, 1989) for “squash” functions, such as the sigmoid function, that map the entire real line to a xed interval. And became known as the *universal approximation theorem*. Later this property was proven to be true for any locally bounded piecewise continuous activation function, if and only if, it is not a polynomial (LESHNO *et al.*, 1993). This result, therefore, includes recti er functions and even threshold functions into the universal approximation theorem.

To illustrate how this approximation can be achieved, an example is presented in Figure 11, where a quadratic function is approximated by an NN with 6 hidden neurons with ReLU activation, and a single output neuron with the identity activation function.

3.2.4 Deeper networks

Deep Learning can be considered to be “the study of models that involve a greater amount of composition of either learned functions or learned concepts than traditional machine learning does” (GOODFELLOW; BENGIO; COURVILLE, 2016).

There is no consensus on how many layers exactly de ne a deep model (SCHMIDHUBER, 2015). Nevertheless, especially knowing that models with a single hidden layer can already approximate any continuous function in a closed interval, one can discuss the effects of depth, that is the advantages using a deeper model.

The universal approximation theorem means that any task done by any neural network could be done by a network with a single hidden layer. However, by hierarchically de ning intricate functions in terms of simpler ones we achieve better representational ef ciency.

A neuron in a deeper layer can reuse computations learned by the previous ones in order

Figure 11 – Approximation of a quadratic function leaned by a neural network with 6 hidden ReLU neurons.

Source: Elaborated by the author.

Description: (a) The original function; (b) shows $z_i^{(1)} \ w_i^{(2)}$ the 6 pre-activation values z of each neuron (already multiplied by the respective weights of the output neuron) as a function of the input x. (c) The strait lines in the lower corners are $h_i^{(1)} \ w_i^{(2)}$, the same functions of the previous frame after passing through the ReLU activation function. All their negative values are pruned to 0 and therefore have no effect on the nal composition $z^{(2)} = \sum_i h_i^{(1)} \ w_i^{(2)}$ represented in pink. Notice that, where their values are non-zero, their slope is added to compose the slope of the nal function, but not where their values are zero. The biases of the hidden neurons are the y intercept of their pre-activation functions $z_i^{(1)}(x) = x \ w_i^{(1)} + b_i$: changing the biases change the x-intercepts, once $z_i^{(1)}(x)$ are linear functions, and therefore move the bend point of $h_i^{(1)}$ in the x axis, changing when $h_i^{(1)}$ starts or stops to contribute to the output. If the hidden neurons didn't have an activation function, every $z_i^{(1)}$ would in uence the output along its entire domain, and any composition of these functions would result in a linear function

to represent its own function, and a function learned by a single neuron can be re-used by many others that come after. This reuse of computation grows exponentially in terms of the depth of the network. In fact, it has been proven that many functions representable by a deep fully connected network can require an exponential number of neurons to be represented by shallow networks (MONTÚFAR, 2014).

A more recent work from Cohen, Sharir and Shashua (2015) showed that almost all functions (probability 1) representable by a Deep Neural Network (DNN) of polynomial size, require exponentially more neurons to be represented, or even approximated, by a shallower network: the increase in the required number of neurons is exponential over the difference of in depth. These results even hold for networks that use convolutions and pooling, like the ones discussed in section 3.4.

Choosing a deep model encodes the assumption that the functions we want the model to learn should be a composition of simpler ones. Empirically, this tends to result in higher generalization (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.2.4.1 Size of the network

The size of the input and output layers are defined by the task being modeled by the network. However, the number of hidden layers, and the number of neurons in each hidden layer affect the capacity of the network. The more neurons in a layer, the more intricate the functions that can be represented by the layer. And the more hidden layers in the network the more levels of compositions of functions it can do.

The ideal number and size of hidden layers will depend on the complexity of the function that the hidden layers must learn for the network to perform the desired task. Also, the higher the capacity of the network, the larger the training set needed for the network to learn functions that generalize to unseen data. Therefore, the number and size of hidden layers is treated as a hyperparameter and must be tuned, manually or with a search procedure, using a validation dataset (AGGARWAL, 2018).

3.3 Optimization

3.3.1 Gradient descent

As mentioned in section 3.1, training consists of looking for a function in a hypothesis space. The hypothesis space of the model is defined by the family of functions $y = f(x; w)$, and is parameterized by w: choosing a different array of parameters w results in a different function.

We want to find the function in the hypothesis space that best perform the desired task, and we estimate this performance over the training set using a criterion that measure the performance of the function implied by the parameters. Usually the problem is framed in terms of a *Loss function*, which measure how bad (according to a formal criterion) a given function performs. If we concatenate all the parameters of the model (including the bias terms) into a single vector w, and consider the sets $X; Y$ containing all training examples $x^{(i)}$ and $y^{(i)}$, then we can represent a criterion by a loss function

$$L(Y; X; w)$$

Let us assume L as given for now. To optimize this function, we can use the *gradient descent* method, which consists of iteratively changing w in the direction that decreases the objective function. The gradient of a function with respect to a variable gives the direction of greatest local increase, but since the loss function is a criterion that measures how bad the model performs, we want to minimize it. So, we change w in the opposing direction of the gradient by subtracting the gradient from w at each iteration.

$$w \leftarrow w - \alpha \nabla_w L(Y; X; w)$$

The factor α in this expression is a hyperparameter that modulates the sizes of the steps in the direction of descent. Note that the gradient is generally different for different configurations of w, so it needs to be recomputed at each step.

Since the size of dataset is often too large and computing the true gradient for the entire dataset is computationally expensive, we can use estimates of the gradient by computing gradients for one random example or one small random subset of the dataset. These approaches are known respectively as *stochastic gradient descent* (SGD) and *minibatch* gradient descent. For these approaches to be possible we need a loss function that can be split in independent terms for each training example in the form

$$L(Y;X;w) = \sum_i L_i\left(y^{(i)};x^{(i)};w\right)$$

Then, since derivatives are distributive over sums, the contribution a single example makes to the gradient can be computed and used in an SGD step

$$w \leftarrow w - \alpha \nabla_w L_i\left(y^{(i)};x^{(i)};w\right)$$

The derivatives with respect to w are computed at for each example using the backpropagation algorithm (RUMELHART; HINTON; WILLIAMS, 1986; WERBOS, 1974), which is an efficient algorithm for computing gradients of deep composition of functions.

3.3.2 Loss functions

During training, the objective is to find parameters that make the output for examples of the training set as close as possible to the target values. The loss function defines how the errors the model makes are penalized. For regression tasks, usually squared errors are used, which is a criterion known as *least squares*.

$$L_i\left(y^{(i)};x^{(i)};w\right) = \left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

When paired with a network of linear units, this results in the exact same model as linear regression.

3.3.2.1 Maximum likelihood principle

For models that predict a probability distribution (or probability density distribution) such as logistic regression, there is a statistical principle for estimating the parameters of the model from a dataset, the *Maximum Likelihood Principle*.

The likelihood of the parameter values *w*, given a set of outcomes, is the probability that such outcome would have to happen in the probability distribution defined by those parameters.

By assuming that each event has been drawn independently from that distribution, the probability of all the events is just the product of all the probabilities.

$$L(w \mid Y; X) = P(Y \mid X; w) = \prod_i P\left(y^{(i)} \mid x^{(i)}; w\right)$$

The maximum likelihood principle states that we should select the parameter with highest likelihood of having generated the training set:

$$w_{ML} = \arg\max_w \prod_i P\left(y^{(i)} \mid x^{(i)}; w\right)$$

In order to be able to compute the contribution of each example to the gradient independently, we can transform this product over all examples in a summation by taking the logarithm of the likelihood:

$$w_{ML} = \arg\max_w \log\left(\prod_i P\left(y^{(i)} \mid x^{(i)}; w\right)\right) = \arg\max_w \sum_i \log P\left(y^{(i)} \mid x^{(i)}; w\right)$$

Since the logarithm is a monotonically increasing function, it does not change the location of the maximum and therefore the argmax of the log likelihood is the same as the argmax of the likelihood. Finally, since the optimization problem is conventionally framed in terms of minimization, we define the loss function to be the negative of the function we are trying to maximize:

$$L = -\sum_i \log P\left(y^{(i)} \mid x^{(i)}; w\right)$$

Now we only need to substitute the probabilities that the model predicts for the true label y_i. In the case of *softmax* output units, the probability is simply the output of the unit that corresponds to the correct class:

$$P\left(y^{(i)} \mid x^{(i)}; w\right) = \hat{y}_j^{(i)}$$

where j is the correct class of $y^{(i)}$. Remember that, in softmax output layers, the output $\hat{y}_j$ is not a label prediction, but the predicted probability of label j being the correct one. The *sigmoid* output unit, on the other hand, uses a single output to predict the probability of two opposing classes, so the predicted probability of the label $y^{(i)}$ is:

$$P\left(y^{(i)} \mid x^{(i)}; w\right) = \begin{cases} \hat{y}^{(i)} & \text{if } y^{(i)} = 1 \\ 1 - \hat{y}^{(i)} & \text{if } y^{(i)} = -1 \end{cases}$$

which can be rewritten as:

$$P\left(y^{(i)} \mid x^{(i)}; w\right) = \frac{y^{(i)}}{2} \quad 0.5 + \hat{y}^{(i)}$$

The least squares criterion for regression has an interesting interpretation under maximum likelihood principle: It is equivalent to modeling $P\left(y^{(i)} \mid x^{(i)}; w\right)$ as a gaussian distribution with mean on $\hat{y} = f(x; w)$ and a fixed variance, and then predicting the value with higher probability, the mean itself (Bishop, 2006).

3.4 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is defined as a network which uses at least one convolutional layer (AGGARWAL, 2018). CNNs are architectures designed to work with grid structured inputs which have strong spatial dependence in the local regions of the grid, such as images. This section describes the operations commonly used in CNNs, convolutions and pooling, and discuss their effects in neural networks and how they can be useful to process image data.

3.4.1 Convolutions and Cross-correlations

Convolutions and Cross-correlations are operations between functions that result in another function. The input functions themselves may be defined over continuous and uncountable domains, like the $\mathbb{R}^n$. For simplicity, the generalized definitions of these operations – which retain all original properties that may be relevant in other application domains – will not be presented here.

The version of these operations most used in digital image processing and CNNs are the 2-dimensional finite discrete convolutions (and cross-correlations), where the functions over finite discrete domains are represented by matrices.

In the finite discrete case, the 2d cross-correlation consist of sliding a matrix W called *kernel* or *filter* over a matrix X, and treating the overlapping regions as vectors and performing a dot product between them. Each possible position of overlapping will produce an entry of the resulting matrix (Illustrated in Figure 12 (a)). For that reason, the cross-correlation is also known as the *sliding dot product*. The convolution does a similar operation, which is equivalent to a cross-correlation with W flipped in every axis – same as rotated by 180 degrees in the case of 2 axes.

Then the *cross-correlation* is defined as:

$$(W \star X)_{i,j} = \sum_{r=1}^{m^{(w)}} \sum_{s=1}^{n^{(w)}} W_{r,s} X_{i+r-1,\, j+s-1}$$

And the *convolution* operation is defined as:

$$(W \quad X)_{i;j} = \sum_{r=1}^{m^{(w)}} \sum_{s=1}^{n^{(w)}} W_{r;s} X_{i+m^{(w)} \quad r;\, j+n^{(w)} \quad s}$$

With the operations de ned in this way, the resulting matrix $H = (W?X)_{i;j}$ is only de ned for positions $i \quad m^{(X)} \quad m^{(w)} + 1$ and $j \quad n^{(X)} \quad n^{(w)} + 1$, therefore, the dot product is performed only in positions where W ts entirely inside X, and the resulting matrix has the size of the maximum i and j. Note that in the case of W and X having equal size, the operations correspond to a single dot product and results in a scalar.

There are situations where it may be desired to preserve the size of X in the output, or to slide the central cell of W (when it has odd dimensions) over the entire X, or to take all possible overlaps between the matrices into account. In such situations the matrix X may be padded with zeros to achieve the desired result. This is illustrated in Figure 12 (b) and (c).

Figure 12 – Illustration of convolutions

Source: https://github.com/vdumoulin/conv_arithmetic

Description: (a) convolution without padding. (b) Convolution in image padded with zeroes to preserve the size of the image in the output, known as *same-padding*. (c) Convolution with padding to allow the kernel to slide across all possible overlapping positions, known as *full-padding*

When being used in CNNs, convolutions and cross-correlations can be viewed as being done over X and parameterized by W. In regions where X has values that are correlated with W (or W rotated in the case of convolution), the resulting dot product will have higher values than in regions of X that are almost orthogonal to W. Therefore, the output H will indicate where the values of X are correlated to W. This behavior can be used to detect features in the input X. For this reason, H is referred to as *feature map* or *activation map* of the lter W.

Convolutions and cross-correlations are commonly used in image processing for spatial ltering and for detecting low level features like points and edges. In this classical approach, however, those operations are done with lters carefully designed by hand to each feature to be detected (GONZALEZ; WOODS, 2008).

The strategy with CNNs is to let the network learn which kernels detect features relevant to the application. Moreover, CNNs normally use many layers of convolutions to take advantage of the hierarchical composition of Deep Learning: The kernels of deeper convolution layers are used to detect features composed of features already detected in the previous layers (AGGARWAL, 2018).

For CNNs, there is no signi cant difference between using convolutions or cross-correlations because a convolution with a lter rotated by 180 degrees will result in the same activation map as a cross correlation. If the network is implemented with one operation, it will learn the same lters as it would with the other (only rotated in 180 degrees) and the outputs of each layer would be the same.

The term convolution became more popularized (hence the name Convolutional Neural Network), but many machine learning libraries implement the cross-correlation operation instead (GOODFELLOW; BENGIO; COURVILLE, 2016). There are even texts that present the CNN architecture with the operation of cross-correlation but call it a convolution (AGGARWAL, 2018).

For the remaining of this text the term convolution will be used but referring to the cross-correlation operation since it may be easier to describe and understand its effects. In an eventual situation where the genuine convolution is necessary, or where choosing between the operations would result in differences other than a rotated W, it will be called to attention.

3.4.2 E ects of convolution layers in neural network

The neural network interpretation for a convolution layer is that every value in the activation map correspond to the output of a neuron connected in a particular way with the previous layer. Instead of connecting to all neurons of the previous layer (or values of the input layer) such neurons are connected only to the position where the dot product with W results in their values. In this interpretation, the weights of the connections are the weights of the kernel W, but replicated across all the neurons of layer H. This impose two restrictions known as local connectivity and parameter sharing. Figure 13 illustrates the neural network interpretation of a 1-d convolution layer.

3.4.2.1 Local connectivity

Having fewer parameters reduce the memory requirements and improves statistical ef ciency of the algorithm (GOODFELLOW; BENGIO; COURVILLE, 2016), which means that the model is less prone to over tting with the same amount of data, or that it may require a smaller dataset to achieve a given degree of generalization. This is because fewer parameters results in a model with fewer admissible functions.

Of course, when regularizing the model, we must include restrictions that will be bene -cial to the task and domain. We want to avoid functions that are spurious or prone to over tting. The functions that tend to perform well should not be avoided by the regularization. The local

Figure 13 – Neural network interpretation of convolutions

Source: Adapted from Goodfellow, Bengio and Courville (2016).

Description: Illustration of the neural network interpretation of a 1-d convolution layer showing the local connectivity and the sparsity of parameters. Each neuron is connected only with the components of the input that are spatially related to it. And the same weights are used in the connections of each neuron

connectivity or spatial dependence of neurons of adjacent layers brings the assumption that the pixel values that compose one feature (or the sub-features that compose a higher-level ones) should occur next to each other.

This assumption can be useful in the domain of images, where pixel values must group in a speci c way to form edges, corners and other low-level features, and where a variety of objects and objects parts, are composed of lower-level features grouped in a speci c spatial con guration (AGGARWAL, 2018).

3.4.2.2 Parameter Sharing

Another property of image data is that its interpretation exhibits a certain level of translation invariance (AGGARWAL, 2018): Usually, a relevant object should be detected regardless of appearing in one side of the image or another. This is an assumption made when imposing parameter sharing with convolutions.

Convolutions are equivariant to translation[1], that is, when the same input is translated by some quantity, the output is the same, but also translated by the same quantity. If a lter learns to detect some feature it will scan the entire previous layer for it, produce high activation where it nds the feature.

Note that this is not con icting with the locality restriction imposed by the local connectivity of neurons. One feature can be detected anywhere, as long as the sub-features that compose

[1] In the strict sense of binary operations between functions, only convolutions are equivariant to translation: If any of the operand functions are translated in their domain, the output function is translated in the same way. Cross-correlations are equivariant only with respect to one of the operands, the function X (when following the de nition on this text). If W is translated, the output of a cross-correlation is translated in the opposing direction. When viewing Convolutions and Cross-correlations as unary operations over input X and parameterized by W, one could say that both are equivariant to translation on the input.

it are arranged in the previous layer in the exact way the filter is expecting.

Like local connectivity, weight sharing also works a regularization on the network. The number of connections between the neurons remain the same, but now they are controlled by fewer parameters which can also reduce overfitting.

3.4.3 Multiple Filters

A filter in a deeper convolution layer can learn to detect a feature composed of lower level features detected in the previous layer. However, complex features are not composed of just one type of lower level features. Therefore, is common to a convolution layer to have many filters. This results in many 2-d activation maps per layer, all stacked in a 3-d tensor, and thus the processing layer itself gains a third dimension, usually referred to as the depth of the layer. This depth of the layer should not be confused with the depth of the network which refers to the number of layers in the network.

The idea is that each filter $W^{(p;k)}$ in layer $H^{(k)}$ will look for a different feature in the image and produce its own feature map in that layer. In the following layer, $H^{(k+1)}$, one filter $W^{(q;k+1)}$ might be looking for a higher-level feature that is composed of more than one type of feature of layer $H^{(k)}$, therefore it needs to consider all activation maps of layer $H^{(k)}$. For this reason, the convolution operation on CNNs for 2-d images uses 3-d tensors as filters instead of 2-d matrices.

Let the 3d tensor $W^{(p;k)}$ be the filter of the convolution producing the p-th activation map on layer $H^{(k)}$. The pre-activation value of the neuron in position $(i;j)$ of such activation map is defined as

$$z^{(k)}_{i;j;p} = \sum_{r=1}^{m^{(p;k)}} \sum_{s=1}^{n^{(p;k)}} \sum_{t=1}^{d^{(k-1)}} w^{(p;k)}_{r;s;t} \, h^{(k-1)}_{i+r-1;j+s-1}$$

The filter doesn't slide across the 3rd dimension, it extends itself across the full depth of the previous layer, considering all activation maps in all dot products. That is because there is no real 3rd spatial dimension in images.

The width and height coordinates of a neuron activation in tensor $H^{(k)}$, correspond to a coordinate in a feature map, and therefore is related with the location of the feature in the image. On the other hand, the depth coordinate is not related with a location in the image, but with which feature map the activation belongs to, and therefore which feature is being measured. Figure 14 illustrates the convolutions operations between consecutive layers.

3.4.4 Pooling

A Pooling layer replaces the output of the network at a certain location with a summary statistic of the nearby outputs (GOODFELLOW; BENGIO; COURVILLE, 2016). The most

Figure 14 – Convolution layers with multiple filters

Source: Adapted from https://commons.wikimedia.org/ - Creative Commons

Description: Illustration of consecutive convolution layers with multiple filters. The highlighted slices of layers $H^{(1)}$ and $H^{(2)}$ are a single activation map produced by one filter. The filters of each layer convolute over the previous one. In this example, the input has only 2 dimensions, and therefore, the filters of layer $H^{(1)}$ have only 2 dimensions as well. However, by having multiple filters, $H^{(1)}$ has multiple activation maps and becomes a 3-d tensor. For that reason, the filters of $H^{(2)}$ must be 3-d tensors with depth equal to the depth of $H^{(1)}$.

commonly pooling is the *max-pooling*, where the maximum value of nearby outputs is used. Other statistics could be used, such as the average, the Euclidean norm; or a weighted average based on the distance to the central pixel. However, these other variants are rarely used (AGGARWAL, 2018).

No pooling is done *across* the depth of the layer. The pooling operation is done over each activation map independently, producing a new activation map for each one in the previous layer. The max-pooling operation is defined as:

$$\text{pool}\left(h^{(k)}_{i,j,p}\right) = \max_{r,s} h^{(k)}_{r,s,p}$$

for (r,s) within a rectangular window around (i,j).

Pooling helps to make the representation approximately invariant to small local translations on the input (GOODFELLOW; BENGIO; COURVILLE, 2016). This is different from the translation equivariance introduced by the parameter sharing of the convolution layer. The shared weights of the convolution layer (caused by sliding the same filter across the entire spatial domain of the previous layer) make it possible to measure a feature wherever it occurs in the image as long as the sub features expected by the filter appear in spatial configuration it expects them to appear.

Pooling introduces a slight translation invariance to the spatial configuration these sub-features need to be on the previous layer for the filter to be able to detect them. That is because, in the new activation map produced by the pooling, neurons will have high activation value just by having the feature occur near them. An illustration of how this can be useful to detect higher level features is presented in Figure 15.

Figure 15 – Illustration of the effect of pooling in Convolutional Neural Networks

Source: Adapted from http://brohrer.github.io/blog.html

Description: (a) An 9x9 image of a white X in black background. The colored rectangles are examples of features that compose the X and the circles mark the locations that would have high activation for a kernel with these exact features. A latter layer could look for activations in these exact places in order to detect the X. (b) In a slightly rotated X, the features are now detected in a different place. If a latter layer is trained to look for activations in the same locations of image a, it will not detect the X in the image. (c) After passing through max-pooling, the activations produced by image b now spread to a neighborhood including the locations of the activations of image a. Therefore, a later layer that would detect an X in image a may also detect it in image c.

3.4.5 Strides

Both convolution and pooling are done by sliding a window across the width and height of the activation maps of the previous layer. In the case of the convolution, such window extends across all the activation maps, considering them at the same time, while the pooling is done for each activation map independently, but they slide across width and height in a similar way.

A positioning of this window defines which values will be taken into account to produce a single scalar for the new layer. It is possible to reduce the size of the output the new layer by skipping some of these window positionings. The distance across each axis between the adjacent window positions to be considered is called the *stride* of the operation. A stride of 1 means considering every window position. When a stride S_k is used in the k-th layer, we consider only the positions starting at location 1, $S_k + 1$, $2S_k + 1$, and so on along both spatial dimensions (AGGARWAL, 2018). An illustration of an operation with stride of 2 is presented in Figure 16.

The use of strides reduces the number of computations that need to be done and the number of activation values that need to be stored during the optimization of the network. Furthermore, CNNs typically have fully connected layers as their final layers, and the number of parameters of these layers will depend on the size of the layer used as input to them. Therefore, strides can both reduce computational cost and memory requirements, and increase statistical efficiency of the model. Strides are commonly used in pooling layers to summarize the responses

Figure 16 – Effect of strides

Source: https://github.com/vdumoulin/conv_arithmetic

Description: Illustration of a convolution or pooling done with a stride of 2. The kernel skips the positionings adjacent to the initial position, resulting in a smaller activation map.

over a whole neighborhood with a single output. (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.5 Autoencoders and representation learning

An auto encoder is a neural network consisting of 2 parts, an **encoder** $f(\cdot)$ that maps inputs to an intermediary representation called the **code** $h = f(x)$, and a **decoder** $g(\cdot)$ that maps the code back to a reconstructed version of the input $\hat{x} = g(h) = g(f(x))$. Each $f(\cdot)$ and $g(\cdot)$ are NN on their own (usually symmetric), and can be deep, nonlinear, and even convolutional. The auto encoder as a whole is trained to make its output $\hat{x}$ as close as possible to the input x. As a result, in order for the decoder to be able to reconstruct the data element, the encoder must produce a feature vector that encodes the features of the element that are relevant with respect to the dataset D, otherwise the decoder would not be able to infer what element to reconstruct.

The auto encoder is trained as a single network $g(f(x)) = \hat{x}$ using a loss function $L(x, \hat{x}) = (x - \hat{x})^2$ based on a reconstruction error that compares the replicated output to the input. The task of replicating the input could be considered supervised learning, since the input itself is being used as the target label. However, the real objective of training an autoencoder is to learn useful representations h, for which we don't have the correct target label. Therefore, autoencoders perform unsupervised representation learning.

To understand how the apparently useless task of learning an identity function can yield to interesting data representations, let us introduce the manifold hypothesis and discus how auto encoders take advantage of it to capture features that other architectures may ignore.

3.5.1 Manifold hypothesis

A manifold is a generalization of curve and surface for higher dimensions: a curve is a 1-d manifold, and a surface is a 2-d manifold. An n-d manifold embedded in m-d space can have any shape as long as it is continuous, and each point (except for boundary points) has a local neighborhood resembling an n-d Euclidean space.

This neighborhood implies the possibility of moving on the manifold along any of these n dimensions, and of identifying a point on the manifold by either its coordinates in m-d space, or by some n-d coordinate system on the manifold. For example, the surface of the earth is a 2-d manifold embedded in 3-d space: we can walk north-south and east-west and refer to a point by latitude and longitude. In the context of machine learning, the concept of manifold is looser, and its dimensionality is allowed to vary from one point to another (GOODFELLOW; BENGIO; COURVILLE, 2016).

The manifold hypothesis is the hypothesis that high dimensional data tend to lie close to a lower dimensional manifold or a collection of manifolds, with interesting factors of variation of the data occurring along directions that lie on these manifolds. This hypothesis is believed to be the case in the context of machine learning tasks involving high-dimensional data such as images, sounds or text, and has been supported empirically for many datasets (GOODFELLOW; BENGIO; COURVILLE, 2016). Assuming this hypothesis, lets define the input domain to the set of all possible inputs x, and the data domain as the subset of the input domain, for which the data generating process of the intended application domain produces valid inputs.

Principal component analysis (PCA) is a technique that finds an orthogonal basis of lower dimension which "best" preserve the data variance. The criterion for best is the following: the first vector of the basis (the first principal component) points in the direction of largest variance of the data. Each new vector added to the basis is the vector that points in the next orthogonal (to all other vectors already in the base) direction with largest variance. With that criterion, vectors are added to the basis until the space spanned by it reaches the desired dimension.

Equivalently, PCA can be seen as finding a lower-dimensional linear manifold. If we project all the data into an l-dimensional linear manifold, we can represent the data by their l-dimensional coordinates in such manifold. The manifold chosen by PCA is the one that lies closest to the data. That is, the positions of the elements in the original space when projected onto the manifold are as close as possible to the original datapoints according to an Euclidean norm. An autoencoder with a single hidden layer containing n linear units and using L_2 loss function spans the same subspace as PCA (GOODFELLOW; BENGIO; COURVILLE, 2016).

Now, since a DNN can approximate any function, it could approximate a function that maps each point to its coordinate in a curve manifold that goes very close to the data points. Still, the function approximated by an NN at the end of training highly depends on the objective function used. In the next section, we analyse the internal representations learned by classifiers in relation with the intrinsic data manifold.

3.5.2 Representations from Classifiers

Let us take a DNN classifier as an example. A DNN classifier can be seen as a linear classifier (the last layer) operating in a transformed space produced by a feature extraction function $\varphi(\ )$ composed of all hidden layers. The hidden layers of a successfully trained NN classifier transform the input extracting features that are relevant for classification. As will be seen in Chapter 4, works on literature take advantage of this fact, and use the hidden layers of a deep NN classifier as a feature extractor to get a higher-level representation of the datapoints.

However, an NN classifier has an objective function that only rewards correct classification, and that may not be the best objective function to get a close representation of all factors of variation on the dataset because the classifier is trained to produce an output that is invariant to factors of variation inside the same class. That is, assuming that different classes lie on different manifolds (or disjoint regions on the same manifold), the output of a classifier should be invariant to factors of variations that correspond to movement on the same manifold (GOODFELLOW; BENGIO; COURVILLE, 2016). This results in higher layers of representation amplifying aspects of the input that are important for discrimination and suppressing the ones that aren't (LECUN; BENGIO; HINTON, 2015).

For an NN classifier to be able to do a good (or even perfect) classification of the dataset, the hidden layers are only required to remap datapoints of different classes to make them linearly separable. The objective function will not reward $\varphi(x)$ for capturing factors of variation that are not necessary for classifying the dataset correctly. Therefore, the internal representation $\varphi(x)$ learned by a classifier may not be sufficient for situations where it is important to capture intra-class variations Figure 17 contains an example of transformations learned by an NN classifier which is able to classify the dataset perfectly, despite not being able to represent the data manifolds perfectly.

3.5.3 Manifold learning with autoencoders

Autoencoders try to learn the manifold structure by learning an intermediary representation from which is possible to reconstruct the training examples. For the decoder to be able to reconstruct the datapoints using only the code, the encoding function must be injective at least over the training examples, that is, different training examples should result in different codes.

On the other hand, to prevent the encoder of learning functions that are not useful, such as exact copy functions, some constraints are imposed on h, and/or on the capacity of the model through regularization strategies, so that the autoencoder cannot afford to represent every possible variation on the entire input domain, only those that are needed to reconstructing the training examples. If the data lies near low dimensional manifolds, this results in representations that implicitly capture a local coordinate system for this manifold: only variations tangent do the manifold will result in changes in h (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figure 17 – Latent space of classi ers

Source: Adapted from http://colah.github.io

Description: Illustration of the transformations done by a small 6-layer network: Layers 1 to 4 each have 2 neurons with tanh activations; layer 5 has a single linear neuron to reduce the representation to 1 dimension; the output layer has a sigmoid classi er node. The hidden layers can be seen as a nonlinear feature extraction function $\phi(x)$ that transforms the input for the linear classi er at the last layer. (a): the chart shows a dense dataset where elements of two classes lie onto 2 separated 1-d curve manifolds. Alternatively, the two manifolds can be seen as 2 disjoint regions of the same manifold. The dashed gray line shows a possible continuation between the two to form a single manifold. This 1-d manifold that goes through all elements of the dataset could be straightened out into 1-d Euclidean space – representing the single factor of variation of the dataset – and all elements of the dataset could be reconstructed from its coordinate in this 1-d space by reverting all transformations. (b): Output of the 4th hidden layer. Instead of straightening the manifolds, the hidden layers folded the classes manifolds against themselves to pull the classes apart, making them linearly separable. (c): the last hidden layer reduces representation down to 1 dimension, mapping 4 distinct regions of each manifold on top of each other to the same region on the 1-d space. If the previous layers had straightened the manifolds, $\phi(\cdot)$ would map each point of the manifolds to a distinct value. However, for an NN classi er to do a perfect classi cation, it's not necessary for $\phi(\cdot)$ to be able to do a perfect representation of the dataset, only that $\phi(\cdot)$ makes the classes linearly separable.

The rst constraint we can impose on the autoencoder is that h must have lower dimension than the input domain, so that $g(h)$ will not be able to span the entire input space. Autoencoders with code dimension lower than the input dimension are known as *undercomplete autoencoders*. However, even not being able to span the entire input space, if the autoencoder has too much capacity, it could learn encodings uncorrelated with the dimensions of the manifolds. Therefore, there are regularizations strategies to control the capacity of the autoencoder and induce it to learn representations that capture the structure of the manifold.

3.5.3.1 Regularizations for autoencoders

Sparse autoencoders are autoencoders whose loss function involves a sparsity penalty over the code layer $L(x;\hat{x}) = (x - \hat{x})^2 + \Omega(h)$. This penalty is grounded on the assumption that most representations should be sparse, that is, that most features are not relevant to representing most of the inputs. And therefore, it would be reasonable to impose that features that can be interpreted as “present” or “absent” should be absent most of the time (GOODFELLOW; BENGIO; COURVILLE, 2016). On way to impose a sparsity penalty is to add the L_1 norm of the h vector to the loss function

$$\Omega(h) = \lambda \sum_i |h_i|$$

Another penalty that can be added to the loss function is a penalty over the derivatives of h with respect to x. This penalty leads the encoder to learn functions that vary slowly for small variations of x, mapping a neighborhood of input points to a smaller neighborhood of codes. Autoencoders trained with such regularization are called *contractive autoencoders*. One commonly used penalty is

$$\Omega(h) = \lambda \sum_i \sum_j \left(\frac{\partial f(x)_i}{\partial x_j} \right)^2$$

The contractive penalty alone would encourage the encoder to learn features that are constant with respect to x. However, regularized autoencoders are trained to satisfy two opposing objectives: the reconstruction error and the regularization penalty. Once the encoder cannot ignore the variations of the data, the intent of using a contractive penalty is that the derivatives of $f(x)$ will be smaller in directions orthogonal to the manifold than in tangent directions (GOODFELLOW; BENGIO; COURVILLE, 2016).

Normally, autoencoders are trained to minimize some loss function $L(x; g(f(x)))$ between the input x and a reconstruction that the autoencoder learns to make from the original input x. *Denoising autoencoders* are trained to reconstruct x from a corrupted version of the input $\tilde{x}$, so the loss function becomes $L(x; g(f(\tilde{x})))$. This corrupted version of the input is typically obtained by adding a small random gaussian noise intended to move the input vector to a small local neighborhood in the input space.

$$\tilde{x} = N(x; \Sigma)$$

With such training, the autoencoder learns to map small perturbations $\tilde{x}$ back to x, right on the manifold. This tends to make the representation learned by the encoder insensitive to variations orthogonal to the manifold (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.5.3.2 Variational Autoencoders

A variational autoencoder (VAE) (KINGMA; WELLING, 2014) differs from vanilla (ordinary) autoencoders in two important ways: First, its encoder and decoder are probabilistic, that is, instead of outputting a fixed value, they model probability distributions $\hat{p}_\theta(z|x) = f(x;\theta)$ and $\hat{p}_\phi(x|z) = g(z;\phi)$, where z are latent factors and f and g are neural networks. The second difference is that VAEs enforce that the latent factors z must follow a given prior distribution by means of a regularization term based on the KL-divergence. To achieve that, the entire network (VAE) is trained to minimize:

$$\min_{\theta;\phi} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{\hat{p}_\theta(z|x)}\left[-\log \hat{p}_\phi(x|z)\right] + D_{KL}(\hat{p}_\theta(z|x) \parallel p(z))\right] \tag{3.1}$$

The first term in equation (3.1) is the negative log likelihood of the model with respect to the data. This term will pressure the model to assign high probability to the data and can be seen as the reconstruction loss. The second term is the KL-divergence between the estimated posterior distribution $\hat{p}_\theta(z|x)$ and the prior distribution $p(z)$. This term can be seen as a regularization loss that pressures the model to arrange its latent variables in a distribution that is close to $p(z)$.[2] That happens because, after taking the expectation with respect to x in equation (3.1), $\hat{p}_\theta(z|x)$ is marginalized into $\hat{p}_\theta(z)$, and the effective KL-divergence measured becomes $D_{KL}(\hat{p}_\theta(z) \| p(z))$.

[2] In the original proposition, the second term was derived as a direct consequence of the original directed graphical model under the variational bayes framework (KINGMA; WELLING, 2014), but the VAE can also be viewed as an probabilistic autoencoder with a KL-divergence regularization loss (HIGGINS et al., 2017).

CHAPTER

4

DEEP LEARNING FOR MEDICAL IMAGE RETRIEVAL

The objective of this dissertation is to investigate Deep Learning techniques that can be used for content-based image retrieval on the medical domain. And as such, it is necessary to justify not only the choice of this scope, but also the choice of the approaches to be investigated – and by consequence the ones to be avoided – to ful l this scope.

Hence, this chapter brings a discussion on the bene ts of CBIR and how it can be useful in Computer-Aided Diagnosis (CAD), as well as a discussion on with approaches may better capture these bene ts. The organization is as follows:

- Section 4.1 illustrates how CBIR is typically used in the context of Computer-Aided Diagnosis (CAD);
- Section 4.2 comments on the potential of Deep Learning for Medical Image Analysis and why to pursue its use in CBMIR;
- Section 4.3 discusses shortcomings of using only class predictions to assist diagnosis;
- Section 4.4 discusses advantages of using similar images to assist diagnosis;
- Section 4.5 discusses disadvantages of using classi cation for retrieving similar images;
- Section 4.6 presents related works and their approaches;
- Section 4.7 discusses the main shortcomings of these approaches and how the use autoencoders could potentially improve them.

4.1 Image retrieval for assisting diagnosis

Ideally, a Medical CBIR system should be integrated to the Picture Archiving and Communication System (PACS) (MÜLLER et al., 2004) and the Hospital Information System (HIS), which contains the Electronical Medical Records (EMR) of all its previous patients. These records contain “a large variety of information, ranging from patient demographics and clinical measurements (age, weight, and blood pressure) to free text reports, test results, and images.” (KUMAR et al., 2013).

CBIR techniques potentially assist decision support when diagnosing medical images by searching large archives for similar images that could help interpreting the image in question (AKGÜL et al., 2011). When the physician is not sure about the diagnosis of a particular case, or wants a second opinion, he/she can execute a QBE with the image in question and receive several similar images from previous patients that match that query.

The physician then looks, between the top-ranked images presented in the results, for the images with visual features that more closely resemble the features that concerned him/her, that is, cases that can be considered similar to the one under analysis. By choosing one of the retrieved images, the physician can explore the entire EMR of the respective patient. With that information, the physician can do interpatient comparisons (KUMAR et al., 2013) and interpret the image from the case at hand based on the retrieved images and their associated records (LI et al., 2018).

4.2 Deep Learning in Medical Image Analysis

Although traditionally Content-Based Medical Image Retrieval (CBMIR) has been relying on hand-crafted features (LI et al., 2018), these methods have some weaknesses which limit the performance of the system (HU et al., 2018). According to Li et al. (2018): hand-crafted features depend on expert knowledge to be designed; are time consuming and expensive; and many of such techniques are designed to speci c medical scenarios and cannot be extended to other domains.

On the other hand, deep techniques only require a training dataset that allows to discover the relevant features (LECUN; BENGIO; HINTON, 2015), and are capable of discovering varied types of features when compared to hand-crafted methods. Today, Convolutional Neural Networks (CNNs) are the most successful type of models for images analysis, and the top performers in most medical image analysis competitions (LITJENS et al., 2017), (HU et al., 2018).

Since CNNs are now the most popular architectures, one could wonder why not to skip one step and perform classi cation with of-the-shelf CNNs to assist diagnosis, or why not to use their classi cation output to perform CBMIR. In the next sections we analyze some shortcomings of these strategies.

4.3 Classi cation for assisting diagnosis

Classi cation is one of the most common use of Deep Learning in medical image analysis, and many other such tasks are related to it (LITJENS et al., 2017). It's also a task commonly performed in the context of CAD and CBMIR systems (RODRÍGUEZ et al., 2016; DOI, 2007; AKGÜL et al., 2011). However, the use of Classi cation in the context of CAD have some shortcomings that may be complemented with similarity search.

When using Deep Learning, a prediction provided by a simple CNN image classi er does not provide the physician with an explanation for the predicted diagnosis, neither the physician can discuss with the classi er and provide new relevant information for the system to consider and update its prediction.

Currently for Deep Leaning strategies – which are the basis for the state-of-the-art in computer vision – it is hard to interpret the reasons for the prediction done by the system, since the semantics of the activations of the intermediary neurons are intricate. Like a black box, the inputs and outputs are known, but the internal representations are not well understood (ANWAR et al., 2018; LITJENS et al., 2017; HU et al., 2018).

The eld of CAD emerged as an alternative for early attempts of automated diagnosis. In this new approach, it is assumed that the output of the computer should be utilized by the physicians, not replace them. The computer output is used as a second opinion, and the physicians make the nal decision (DOI, 2007; TAKAHASHI; KAJIKAWA, 2017).

However, classi cation results might not be the best output for assisting the physician's decision. Similar images are often more helpful as the physician “must still judge the retrieved cases and the reasons for retrieving the images are often clearer whereas classi cation results are sometimes hard to detail and need to be explained” (MÜLLER et al., 2004).

4.4 Bene ts of image retrieval for aiding diagnosis

4.4.1 Visual support for decision

Class predictions alone may not be the best output for assisting the physician's decision, especially when it's dif cult to interpret the reasons for the prediction. After reviewing several CAD systems, Doi (2007) suggested that classi er-based systems, when discriminating between malignant and benign cases, should integrate similarity search to display similar cases from each class to increase the con dence of physicians and the accuracy of their decisions. Figure 18 illustrates how similarity search can be used to assist a diagnosis.

Figure 18 – Example of similarity search on Computer-Aided Diagnosis

Source: Doi (2007).

Description: Example of similarity search being used to assist the diagnosis of a query nodule of unknown malignance. By comparing the query nodule with both malignant and benign nodules it is easier to identify which features the query nodule has in common with malignant nodules but not with benign ones. In this example, most observers were able to identify the unknown case correctly as being more similar to malignant masses than to benign ones.

Such visual support with similar images could be very useful to the decision process of the physician because "diagnostic decision making has traditionally involved evidence from the patient's data coupled with the physician's prior experience of similar cases" (KUMAR et al., 2013). Also, just presenting similar diagnosed images, even without predicting the class of the similar one, has already been demonstrated to signi cantly improve radiologists' performance (LI et al., 2003).

4.4.2 Reliability of labels

The output of a classi cation algorithm is subject to error, which can be detrimental in cases where the user is not con dent on whether or not the system is correct. On the other hand, a CBIR system, when integrated with a PACS, retrieves images from a database of already diagnosed images and the reports attached to the retrieved images were made by human physicians. Some of these diagnoses may have even been con rmed by additional laboratorial exams or by the evolution of the patient's condition, providing high degree of clinical con dence.

Even if the system doesn't perform well, the only consequence is that images from cases not so similar will be returned. In such situation the users can judge by themselves if the returned images are similar enough for comparing diagnosis. Differently from classi ers, with CBMIR systems, whatever the performance of the retrieval, the user can have con dence that all the retrieved diagnoses were done by a quali ed physician.

4.4.3 Contextual support for decision

Image classi ers base their decisions solely on image features that they correlate statistically with the labels on their training set, whereas medical professionals can base their decisions on stablished clinical practices and recent scienti c discoveries. Even within a particular case, useful information is not just contained within the images themselves. Physicians often leverage a wealth of information on the patient's clinical history and demographics to arrive at better decisions (LITJENS et al., 2017).

As mentioned in section 4.1, the information attached to the retrieved images may be richer than just the diagnosed conditions. There may be much information (in structured form or in textual reports) that were relevant to such diagnose and can be used for comparison with the case at hand, information such as:

- The received diagnosis and the professional who did it;
- The reason for the diagnosis based on an analysis of the imaging exam;
- Visual annotations on the image itself;
- Other information that may have contributed to the conclusion of the physician such as age, sex, other diseases, family history of risk contributing factors, tried treatments and medications, and laboratorial exams.

It is important to notice that providing an explanation for the prediction is not merely about offering a convenience or about convincing the physicians of the system's ability. In the medical domain, the diagnosis is always a burden of the physicians and they are ethically and legally responsible for their decisions. Therefore, they have to know the reasons of their decision. In such scenario, it is often not suf cient for the system to produce a good prediction, it has to articulate the reasons for the prediction in some way (LITJENS et al., 2017).

With similarity search, the professional has access to records of other patients with similar cases and the motivations for the diagnoses of such patients. Also, the professionals themselves get to decide: (1) which information is relevant; (2) if the cases are similar enough for such information to apply to the patient in consideration; (3) and how the cases correlate to the one being analyzed. Finally, the professional himself makes the diagnosis. The system is not supposed to diagnose the patient, but only provides information that may contribute to the decision process of the professional.

4.5 Image retrieval by classi cation

Having stablished that retrieving similar cases to the one in consideration can have advantages to simply predicting the correct class of the image. Another strategy that can be thought to take advantage of existing CNN classifiers is to use their class predictions to retrieve similar images. This section discusses some problems with such strategy.

Consider a scenario were a radiologist is uncertain about a nodule being benign or malignant. Suppose we want to use a perfect classifier to find images useful to assisting his/her decision. The images of the database are already labeled as benign or malignant. Figure 19 represents a space composed of two visual features (j_1 and j_2) that are highly correlated with nodule malignancy, and normally used by physicians as a visual cue to diagnose malignant nodules.

The hypothetical classifier may not internally represent the domain in those exact axes, but its internal representation is also highly correlated with malignancy, and it produces perfectly accurate predictions. The continuum elliptical curves on Figure 19 are the contour lines of the degree of certainty of the classifier that some example is malignant. The green region is considered by the classifier as benign, and the red one is considered as malignant.

The particular image the radiologist is uncertain about landed in the query point Q. Being a difficult case, point Q is near the decision boundary, which is typical of cases that let radiologists uncertain. The radiologist doesn't know for certain if the nodule in image is malignant and wouldn't like to trust the classifier without a relevant visual or contextual support for the decision, after all, in real world applications, classifiers can make mistakes.

Figure 19 – Hypothetical latent space

Source: Elaborated by the author.

Description: Illustration of a hypothetical scenario of using a classifier CBMIR. The j_1 and j_2 axes represent visual features correlated with nodule malignancy. Point Q represents the query images. Points P1 to P4 are examples of images from the database. The continuum elliptical curves are the contour lines of the degree of certainty of the classifier that some example is malignant. The green region is considered by the classifier as benign, and the red one is considered as malignant.

Consider the hypothetical images whose features lie on points I_1 to I_4 on the plane, and which of those are desired to be retrieved before the others for visual support for diagnosis. We have some options on how to use the output of the classi er to retrieve similar images. We can:

1. Predict the class of the query image and retrieve any image from the same class in no particular order. That is, lter the database by the predicted class.

2. Return images with the highest level of certainty of belonging to the same class as the predicted class of the query image.

3. Return images with the smallest distance to the level of malignancy predicted for the query image.

The rst approach would return any images whose features lie in any part of the red region; the second approach would return I_2 rst; and the third approach would give priority to images that lie close to the dashed line – which represents the same level of certainty as que query point – so I_3 would be returned before the others.

It is clear that using the output of the classi er might not be the best approach as the most relevant image to compare to the query one is I_1 (and perhaps I_4 also if it is desired to do a contrastive analysis as in gure 18). As observed by (AKGÜL et al., 2011), despite the possibility of using the membership to classes as features, similarity measurement should not be viewed as a classi cation task.

The ideal approach – approach 4 – would be not to use the output of a classi er but have features j_1 and j_2 measured for each image and return the images that lie closest to Q in such space. That way, I_1 and I_2 would be retrieved before the others.

Note however that, given that our hypothetical classi er makes perfect predictions, if relevance of retrieval is measured by class label, approaches 1 and 2 would result in a perfect PR-curve (with mAP=1) despite don't providing the best retrieval in a qualitative point of view. On the other hand, despite being qualitatively superior, approach 4 would result in a sub-optimal PR-curve, as it would retrieve I_4 before I_2 and I_3.

That is because, in order to do a qualitative evaluation of retrieval performance, usually class labels are used as relevance criterion. However, it has to be kept in mind that a CBMIR system is not mainly being employed for classi cation, but for nding similar images or cases. Using only class as a criterion for relevance is a simpli cation that is not suitable to evaluating retrieval performance in ne-grained levels (MÜLLER et al., 2004; LI et al., 2018). Also, in a scenario where only images from the same class are desired, this exact procedure can be done while ltering out undesired classes, resulting in similar images from the same class (AKGÜL et al., 2011).

The majority of the approaches found in literature review for the present project use internal representations of classi ers as features for comparing similarity, in the intent that these internal representations would capture the features relevant to the domain, like the hypothetical ones in Fig 19. Such approaches are presented int the next section.

4.6 Current Approaches

This section presents current techniques for integrating Deep Learning in CBMIR that were found during literature review for this project.

4.6.1 Features from pre-trained classi ers

The simplest way to incorporate of the shelf Deep Learning in CBMIR is to use a model already trained in another domain as a global feature extractor. In Bressan et al. (2018), the authors compared the use of the activation values of the last convolutional layers of 3 CNN architectures as descriptors to several traditional hand-crafted descriptors for the task of retrieving similar ROIs of mammograms. They tested the descriptors using different distance metrics and an interactive query re nement technique and evaluated them with the PR-curve. The CNNs had already been trained for the classi cation task on natural images and were not ne-tuned or retrained on the mammogram datasets, and yet, the CNN features performed better than the hand-crafted ones.

In Shah et al. (2016), the authors used a pre-trained CNN as part of a pipeline for a descriptor for patches of prostate Magnetic Resonance image volumes. They took activations of the penultimate layer (fully connected), which had 1000 nodes, then used a random forest of oblique-split trees to transform the 1000-d descriptor of each image patch in a binary hash code. Finally, they used a histogram of the hash bits to combine the hashing codes of various paths into a single feature-vector for an image volume.

4.6.2 Database ltering by class

In Khatami et al. (2018b), the authors trained a CNN classi er in a diverse medical image collection to discriminate between image modality, body part and orientation. During the query by example, they use such CNN to predict the class of the query image and discard all images of different classes. The images of the predicted class are then compared to the query image and ranked using Local Binary Patterns (LBP) and Radon features. The system is evaluated by an error measure designed for evaluating classi cations of this dataset. For that, they consider the class of the rst returned image for each query. The authors report that the system surpasses several state-of-the-art methods in the same dataset.

In another work (KHATAMI et al., 2018a) with the same dataset and same evaluation method as the previous one, the authors train 3 CNNs of different architectures to classify the dataset. During the query, each CNN predicts the two most probable classes, resulting in a set of

2 to 6 probable classes. Then, all images belonging to one of those classes are compared to the query image with LBP, Histogram of Oriented Gradients and Radon descriptors to nd the most similar one.

In Ibanez et al. (2017), the authors trained a CNN on a dataset of lung nodules - manually segmented and annotated by experts - to classify between 5 malignancy levels. They tested the activations of each layer as descriptors for similarity search and settled for the classi cation layer as feature vector for presenting better results.

4.6.3 Features of classi ers trained on domain

In Qiu et al. (2017), the authors took a CNN pre-trained on a natural images dataset, included a new untrained hashing layer before the output layer, and netuned the last two layers of the network to classify between 4 classes: brain, lung pancreas and bladder, forcing the hashing layer to create a binary representation of high-level semantics. The hash code is used to group similar images in buckets of a hash table. During the query the set of candidate images is composed of all the buckets with hash code within a certain hamming distance of the query image's hash code. The candidate images are then compared to the query one and ranked using the activations of two layers: the output layer and the one before the hashing layer.

In Qayyum et al. (2017), the authors trained a CNN classi er to discriminate images from different modalities between 24 classes, each class being a different body part, and used the activations of the last 3 layers before the output layer as descriptors. They also did an experiment where they ltered the dataset by the predicted class and evaluated both scenarios (with and without ltering by the predicted class) using PR curve and mAP.

4.6.4 Regression of ground-truth distances

In section 2.2 it was argued that one ideal ground-truth information to evaluate an image descriptor system would be an all-to-all distance matrix with the ideal distances between the images of the dataset. In fact, such matrix would also be the ideal supervision to train an end-to-end Deep Learning image comparator. Instead of measuring distances between feature vectors learnt for other task (such as classi cation), one could train a regressor to learn a function from the domain of pair of images to real numbers.

That was one of the strategies used in Muramatsu et al. (2018). In this work, the authors have asked eight physicians to produce each an all-to-all distance matrix of rectangular Regions Of Interest (ROI) of mammograms and used the average of this matrices to train a CBIR system using two strategies: In the rst one, they use Multi-Dimensional Scaling (MDS) to learn coordinates for each image of the training set in a new latent space, such that their distances are as close as possible to the ones in the ground-truth distance matrix. Then they train a CNN to learn a mapping from the image domain to this latent space using the new coordinates as labels.

In the second strategy, they train a Siamese network to learn the mapping from pair of images directly to their distances using the average subjective distances as labels.

The dif culty with strategies like the aforementioned work is in producing such a distance matrix. In addition to the high subjectivity and variability in judging similarity or distance, the similarity matrix grows with the square of the training set. In the aforementioned work, the authors had the physicians compare only 27 images which required each physician to evaluate 351 unique pairs of images. Doing this for a big dataset representative of real-world scenarios is infeasible. In that work, due to the small labeled dataset, the authors used heavy data augmentation and needed to evaluate the system with leave-one-out cross validation.

4.6.5 Latent space optimization

One strategy to learn a mapping to a latent space without having an all-to-all distance matrix or ground-truth coordinates in such space, is to use an objective function that approximate the coordinates of objects with the same classes.

In Conjeti et al. (2017), the authors trained a CNN architecture to learn a mapping from bags of a varying number of ROIs of a single medical case to a binary code representation. To aggregate a varying number of images into a single representation, they used a max-pooling layer after the last fully connected one, aggregating the activations obtained by all images of ROIs. This pooling layer is followed by a tanh hashing layer to produce binary representations. This whole architecture is trained with backpropagation one pair of bags at a time using an objective function that penalizes in proportion to the distance between the hashes of the pair, but only if the pair belongs to the same class.

In Chen et al. (2018), the authors also trained a CNN to learn binary hash coding, but the dataset they used had intersecting classes, where a single lung RX may present more than one medical nding. So, they used the intersecting classes as an advantage and used a custom triplet loss that takes into account the number of classes the images share. Additionally, they added a classi cation layer after the hashing layer to train both a descriptor and a classi er at the same time.

4.7 Chapter Conclusions

Most of current approaches discussed in section 4.6 rely on Convolutional Neural Networks (CNNs) trained to perform image classi cation (a task so commonly associated with such architecture that is usually implied by its use) or else trained in more elaborated ways that also rely on hit rate over a certain class of interest. Models trained in such way suffer of two important disadvantages.

The rst disadvantage is that those models require a large amount of accurate and structured annotations for class supervision. This is a challenge for the medical domain because these annotations can only be provided by certi ed physicians specialized at the domain in question, which is expensive. Often hospitals have databases with large amounts of image data of previous patients, but there is no structured annotation available. Such databases could be used for CBMIR once we have the descriptors, but to train descriptors using the classi cation approach, they rst would have to be annotated by a committee in an arduous and expensive process. Because of this, unsupervised methods for generating descriptors with performance close to that of supervised models would be highly valuable for CBMIR.

The second disadvantage of networks trained in classi cation is that they tend to learn intermediate vector representations that capture features that are important for the discrimination of the classes they were trained on, but ignore the ones that are not (LECUN; BENGIO; HINTON, 2015). This phenomenon is discussed in section 3.5.2 and illustrated in Figure 17.

One could claim that the features suf cient for discriminating a set of classes would necessarily be suf cient for a CBMIR system if its purpose was to aid diagnose over those same classes, but this may not be the case: Images in the medical domain can have high intra-class variation, that is, they can vary in many aspects that do not in uence how they are classi ed but have signi cant visual impact over the image (LI et al., 2018; YU et al., 2017). The output of a classi er should be invariant to those visual variations that do not interfere in the class of the image, but in CBMIR it is not enough that the retrieved images belong to the same class of the queried one, they must also be visually similar and, therefore, it is desirable that the descriptors also capture features of intra-class variations (LI et al., 2018; MÜLLER et al., 2004).

The relevance of intra-class variations for CBMIR is illustrated in Figure 20. Since medical images usually have high intra-class variance, ignoring intra-class variations may result in a loss of visual similarity in retrieval. Note that visual similarity is relevant for the comparative analysis that brings the bene ts of CBMIR over pure classi cation for CAD, which were discussed in section 4.4.

Intra-class variations are also related to the shortcomings of using class predictions for CBMIR, discussed in 4.5: Taking again the hypothetical scenario of Figure 19, imagine that instead of using classi cation output of the hypothetical classi er, we instead use its internal representations as features for CBMIR. If those representations are indeed invariant to intra-class variation, they will not span a space similar to f_1 and f_2. Instead, they will all be orthogonal to the contour lines, since movement parallel to the contour lines represents intra-class variation. Hence, the approach of retrieval using internal features from classi ers would suffer from similar problems as using the classi cation output as features for retrieval.

On the other hand, Variational AutoEncoders (sec. 3.5.3.2) are trained without labels and had been shown to produce a latent space that discovers and disentangles intrinsic factors of variation from the data (HIGGINS et al., 2017). Autoencoders have already been suggested for retrieval of natural images (KRIZHEVSKY; HINTON, 2011), but despite their applicability

Figure 20 – Hypothetical CBMIR queries

Source: Elaborated by the author.

Description:(a) is a query image diagnosed as malignant, and (b) and (c) are retrievals for hypothetical descriptors, where the retrieved images in both scenarios pertain to the same class as the queried one. Retrieval (b) is produced by a descriptor whose features don't capture intra-class variations, and therefore makes so a pair of images have similar feature vectors just by virtue of them belonging to the same class. Retrieval (c) is produced by a descriptor whose features do capture intra-class variations, hence, for its feature vectors to be close it is not enough that they belong to the same class, they must also be similar in many factors of intra-class variations. Similarly, while result (c) is more similar to the query image, the retrieved images in both scenarios pertain to the same class as the queried one, and therefore would score the same on a metric based on class hit.

for image retrieval, no works analyzing its use for medical image retrieval were found during literature review. Hence, In the next section, we propose an approach based on VAEs for training descriptors for CBMIR that can take advantage of large hospital datasets that lack annotation, as well as take advantage of those features ignored by classi ers to be more sensitive to intra-class variation. Finally, we show that this approach can yield better results than the ones based solely on classi cation, and can even be used in combination to improve the results of the latter.

CHAPTER 5

VARIATIONAL AUTOENCODERS FOR MEDICAL IMAGE RETRIEVAL

5.1 Proposed solution

In the previous chapter we saw that current approaches for Medical Image Retrieval are mostly based on first training a neural network to discriminate classes from the intended domain and then using the internal representations of the network as features for similarity search. This strategy 1- requires an expensive labeling procedure and 2- can ignore features that, despite being irrelevant for classification, could be important for CBMIR (Section 3.5.2 and Section 4.7).

In this chapter, we propose an approach based on Variational Autoencoders for training descriptors for CBMIR that can take advantage of large hospital datasets that lack annotation, as well as take advantage of those features ignored by classifiers to be more sensitive to intra-class variation. Finally, we show that this approach can yield better results than the ones based solely on classification, and can even be used in combination to improve the results of the latter.

5.1.1 Model

To train a descriptor for CBMIR unsupervisedly, we propose a Variational Autoencoder (sec.3.5.3.2) with a prior $p(z) = \mathcal{N}(0, I)$ on the latent codes. That choice of prior pressures the dimensions of $\hat{p}_\theta(z)$ to have the same scale and to be statistically independent, both properties that are convenient for our application and follow directly from the normal distribution. Then, we model $\hat{p}_\theta(z|x)$ as a diagonal-covariance Gaussian so that the estimated marginal $\hat{p}_\theta(z)$ can be matched to the prior p(z) via the KL-Divergence loss. We can do this by training the encoder f to output predictions for the mean and variance vectors $\left(\hat{\mu}, \hat{\sigma}^2\right) = f(x; \theta)$, so that $\hat{p}_\theta(z|x) = \mathcal{N}\left(\hat{\mu}, \hat{\sigma}^2 I\right)$.

To make the whole network differentiable with respect to the parameters θ and ϕ, we do the reparametrization trick (KINGMA; WELLING, 2014) on the normal distribution. That is, instead of sampling $\hat{z} \sim \hat{p}_\theta(z|x)$ directly, we compute $\hat{z}$ from the deterministic outputs of the encoder $\hat{\mu}, \hat{\sigma}^2 = f(x;\theta)$ with the help of a noise variable $\epsilon \sim \mathcal{N}(0, I)$ and element-wise multiplication: $\hat{z} = \hat{\mu} + \hat{\sigma} \odot \epsilon$.

For the decoder output, we model $\hat{p}_\phi(x|z)$ as a gaussian of constant variance so that its expected value $\hat{x}$ is always the mean. Then the reconstruction term becomes the least-squares loss, and equation (3.1) becomes:

$$\min_{\theta;\phi} \mathbb{E}_{x \sim D}\left[\mathbb{E}_{\hat{p}_\theta(z|x)}\left[\frac{1}{2}(\hat{x} - x)^2\right] + D_{KL}\left(\mathcal{N}\left(\hat{\mu}, \hat{\sigma}^2 I\right) \| \mathcal{N}(0, I)\right)\right] \tag{5.1}$$

And we can get the solution for the Divergence between the gaussians from Kingma and Welling (2014):

$$D_{KL}\left(\mathcal{N}\left(\hat{\mu}, \hat{\sigma}^2 I\right) \| \mathcal{N}(0, I)\right) = -\frac{1}{2}\sum_{j}^{J}\left(1 + \log \hat{\sigma}_j^2 - \hat{\mu}_j^2 - \hat{\sigma}_j^2\right) \tag{5.2}$$

where J is the dimensionality of the gaussians. After the network is trained, we can use the encoder to infer feature vectors for new data as the expected value of the predicted distribution:

$$\hat{z} := \mathbb{E}_{\hat{p}_\theta(z|x)}[\hat{p}_\theta(z|x)] = \hat{\mu} \tag{5.3}$$

5.1.2 Architecture

For the encoder and decoder, we used a simple architecture inspired from the discriminator and generator of ProGAN (KARRAS et al., 2018), but with 3 convolutions per block instead of 2, and an input resolution of 128×128. We used the normalization technique described on that work as “Equalized Learning Rate”, but we have not grown the networks progressively, training instead all the layers at once.

We also added 4 fully connected layers at the end of the encoder and the beginning of the decoder, which performs a nonlinear transformation on the space similar to the mapping network on the generator of StyleGAN (KARRAS; LAINE; AILA, 2018). This allows the uncoupling of each dimension of the code vector from the spatial position it originally occupied on the feature maps, and compresses the code to 128 dimensions. Table 2 describes the architectures of the encoder and decoder .

5.2 Baseline, unsupervised, and mixed models

To compare our approach to the more common approach of using classifiers, we trained a baseline classifier CNN with which to compare our model. In order to ensure fairness of comparison, those networks were constructed in such way that the encoder part of their architecture

Table 2 – Network architecture

Encoder $f( )$		
Layer	Act.	Out. Shape
Image	ReLU	128x128x3
Conv 1x1	ReLU	128x128x16
Conv 3x3	ReLU	128x128x16
Conv 3x3	ReLU	128x128x16
Conv 3x3	ReLU	128x128x32
Max_Pool	-	64x64x32
Conv 3x3	ReLU	64x64x32
Conv 3x3	ReLU	64x64x32
Conv 3x3	ReLU	64x64x64
Max_Pool	-	32x32x64
Conv 3x3	ReLU	32x32x64
Conv 3x3	ReLU	32x32x64
Conv 3x3	ReLU	32x32x128
Max_Pool	-	16x16x128
Conv 3x3	ReLU	16x16x128
Conv 3x3	ReLU	16x16x128
Conv 3x3	ReLU	16x16x128
Max_Pool	-	8x8x128
Conv 3x3	ReLU	8x8x128
Conv 3x3	ReLU	8x8x128
Conv 3x3	ReLU	8x8x128
Max_Pool	-	4x4x128
Flatten	-	2048
Dense	ReLU	512
Dense	ReLU	512
Dense	ReLU	256
Dense	-	128+128 ($[\hat{m}, \hat{s}]$)
Sampling	-	128

Decoder $g( )$		
Layer	Act.	Out. Shape
Code	-	128
Dense	ReLU	256
Dense	ReLU	512
Dense	ReLU	512
Dense	ReLU	2048
Reshape	-	4x4x128
UpSample	-	8x8x128
Conv 3x3	ReLU	8x8x128
Conv 3x3	ReLU	8x8x128
Conv 3x3	ReLU	8x8x128
UpSample	-	16x16x128
Conv 3x3	ReLU	16x16x128
Conv 3x3	ReLU	16x16x128
Conv 3x3	ReLU	16x16x128
UpSample	-	32x32x128
Conv 3x3	ReLU	32x32x64
Conv 3x3	ReLU	32x32x64
Conv 3x3	ReLU	32x32x64
UpSample	-	64x64x64
Conv 3x3	ReLU	64x64x32
Conv 3x3	ReLU	64x64x32
Conv 3x3	ReLU	64x64x32
UpSample	-	128x128x32
Conv 3x3	ReLU	128x128x16
Conv 3x3	ReLU	128x128x16
Conv 3x3	ReLU	128x128x16
Conv 1x1	ReLU	128x128x3

Source: Elaborated by the author.

were identical. The CNN consists of the exact same encoder $f( )$ as the VAE (see Figure 21) with the single addition of a linear classifier $h( )$: a linear layer with sigmoid activation and trained with the cross-entropy loss.

To evaluate how much the VAE would improve the representations of a classifier if used together with it rather than as an alternative, we also trained a mixed model: an architecture identical to the VAE except for the addition of the same classifying head as the CNN stemming from its bottleneck. We train this network with the losses of both the VAE and the CNN, so as to reconstruct and classify the data, and refer to it as supervised VAE (SVAE). Figure 21 describes all those models.

With respect to computational efficiency, the cost of the classifying head $h( )$ and the

Figure 21 – Diagram of the models

Source: Elaborated by the author.

Description: The rectangles are vectors, the trapezes are neural networks. For the training, The SVAE uses all modules, the VAE uses $f(\cdot)$ and $g(\cdot)$, and the baseline CNN uses $f(\cdot)$ and $h(\cdot)$. For inference, all models use only $f(\cdot)$ to compute the feature vector as $\hat{z} = \hat{\mu} = f(x)$, discarding the computation of $\hat{\sigma}$. That way we ensure the encoders of all three approaches have exacly the same architecture, and therefore, the same set of admissible functions.

sampling of z from the parameters μ and σ are all negligible with respect to the size of the encoder $f(\cdot)$ and decoder $g(\cdot)$, which are computationally symmetric. Hence, each training step of the VAE and the SVAE have double the computational cost and memory footprint of the CNN. However, this difference only affects the training procedure. After training, at inference time, only the encoder is used to generate features $\hat{z} = \hat{\mu} = f(x)$, discarding the few operations used to produce σ which is only needed for the probabilistic training. That way, the encoders of all three approaches are identical with respect to computational complexity, and will cost the same to generate features for queries when in production.

Moreover, despite VAE and SVAE having twice as many operations and learned parameters as CNN, their resulting encoder will have the same statistical capacity, number of parameters, and set of admissible functions as that of the CNN, making the comparison between the approaches as fair as possible. In other words, the baseline CNN, the VAE and the SVAE can be seen as three different approaches for training the same encoder $f(\cdot)$ to be a image descriptor.

5.3 Experiments methodology

5.3.1 Evaluation metrics

The usual way to evaluate and compare efficacy of CBIR systems is to compute the Precision-Recall (PR) curve or the mean Average Precision (mAP) of their rankings over a dataset from the intended domain, using “whether or not two images are from the same class” as the similarity criterion. This practice is common because we intuitively expect that two images from the same class have higher probability of being semantically similar than images from different classes.

It also has the bene t that we are measuring similarity with a criterion that is highly semantic, rather than using pixel-level low-semantic metrics. Such domain speci c criteria such as classes are more dif cult to bias in favor of a particular technique. For example, the pixel-wise RMSE is a common metric for similarity, used by the research communities of many multimedia applications. However, if we used RMSE as a metric for CBMIR, a system using a pixel-wise L_2 distance as its ranking criterion would be highly (and unfairly) favored by such metric.

Nevertheless, despite those bene ts and the wide adoption across the CBIR community, we must remember that “class hit” is a surrogate metric for similarity, and, in many applications, those will not be equivalent, certainly not in the case of CBMIR for Computer-Aided Diagnosis. As an example, Figure 20 shows two retrievals that would be equally scored by a class hit metric, but are clearly not equally similar to the query image.

Usually, the coarseness of such criterion would not be a problem when comparing two CBIR systems, since it would affect both of them. However, when comparing an unsupervised system with a supervised one, training the supervised system in the very same class used by a surrogate metric would render the metric heavily biased in its favor.

At rst glance, such bias could be perceived as being simply the usual advantage of supervised methods: We want the system to learn task A and, therefore, we give it examples of task A. If it mimics the examples and perform well on the A-metric, it is because it's doing what we want, not because of metric bias.

But that's not the case here. CBMIR is usually evaluated with such an A-metric (class hit) but, in reality, we want to produce a system that is good at task B (ranking images according to similarity). It is this use of a surrogate metric that opens a vulnerability for evaluation bias.

To circumvent this problem, in this work, we evaluate the supervised and unsupervised approaches with the usual metrics, but using labelled attributes other than the class used to train the supervised model. We call these attributes hidden classes. In contrast, we refer to the class which the system was trained on as `training class', and we also provide the results of the same metrics over it. To show that metrics over hidden classes provide a better description of the ability of a CBIR system in nding similar images than metrics over training classes, we also provide samples of random queries for each model.

5.3.2 Training and Benchmarking procedure

The three models described in section 5.2 were trained on a random training split of 95% of each dataset. The models with classifying heads received supervision for a single class of the dataset. Then, we used the encoders of each network to extract features from the validation split of the remaining 5% of the data to perform the CBIR benchmark. The validation split was not used for any hyperparameter tuning, nor was it used to interfere in training in any way, with the single exception of deciding the stopping epoch on the ISIC dataset. The benchmark was done by performing similarity queries with each element of the validation split over the remaining of the validation elements (without the query element), and computing the mAP and P@10 for the training class and hidden classes.

5.3.3 Datasets

5.3.3.1 Celeba

The rst dataset used for the experiments is the CelebA dataset (Liu et al., 2015). It consists of 200k faces of celebrities “in the wild” – meaning taken from organic natural press photos found on the internet – that have already been aligned and cropped, and each image has ground-truth labels for 40 binary attributes.

We include this dataset of faces in this work for two reasons. The rst reason is the quantitative advantages in training and evaluation: it is a big dataset and some of its classes are well-balanced, allowing us to train Deep Learning models for many epochs without over tting. This allows us to compare the competing approaches in an ideal training setting, and also to remove the hypothesis that any result is due to insuf cient training data. Also, by having all images labeled in 40 binary attributes, it allows us to evaluate the retrieval performance of the models over many hidden classes. The second reason for using this dataset is that human faces are a common domain of expertise, allowing researchers beyond medical specialist to evaluate the results qualitatively.

We trained the models for 500 epochs since all three were able to converge on this dataset without over tting. The attribute chosen to train this network was the sex of the person in the image. This attribute was chosen because it is fairly balanced, having 40% male incidence, and is related to many features of face, hair and clothing, giving the conditions necessary for the baseline CNN to learn rich latent spaces. Also, since there are people with more than one photo in this dataset, we made sure no individual had photos on both the training and validation split.

5.3.3.2 ISIC

To demonstrate the effectiveness of the proposed approach for CBMIR we select the ISIC 2019 dataset, the same dataset used in the hypothetical example in Figure 20. It is a dataset from a competition on identifying skin diseases, with 24k images labeled on 8 classes from which we take melanoma as the training class and use the others as hidden classes for evaluation.

The melanoma class, as most diseases in medical datasets, is underrepresented. In a production environment, a classi er would usually be trained with the aid of a loss that balances sensitivity and speci city over the training class. So again, in order to represent a fair scenario for the baseline, we scale the weight of each training element on the loss based on whether or not it belongs to the training class.

The weight factors for positive and negative elements respectively are $w_P = \frac{|S|}{2T_P}$ and $w_N = \frac{|S|}{2T_N}$, where $|S|$ is the size of the dataset, and T_P and T_N are the number of positive and negative elements respectively on the dataset. We don't use loss balancing for the hidden classes since they are used to simulate attributes for which we wouldn't have any labels.

Since the VAE uses the entire image as the training signal, it is more resilient to over tting, and so it was trained for 1000 epochs for its learning curves to converge. The CNN and SVAE on the other hand have a classifying head which receives a single bit as supervision, and so they over t more easily on datasets of this size. Thus, once again, to fairly represent the supervised approaches, the validation set was used to choose the epoch with best validation accuracy for the CNN and best validation loss for the SVAE, and those were the models used for benchmarking.

5.4 Results and discussion

5.4.1 Classi cation performance check

The rst step in the evaluation was to ensure the supervised models had been well trained in the classi cation task. The classi cation performance over the validation datasets is presented in Table 3 along with reference scores for the data sets.

We can see that the baseline CNN and the SVAE fell short of the reference scores for the ISIC dataset, taken from the best ranked single-network approach (no model ensembles) submitted to the challenge, described as “Densenet-161 with heavy use of random crops” in the challenge leaderboard (ISIC Challenge, 2019). This model reached 0.892 ROC-AUC over the melanoma class, but the authors made use of various techniques for architectural search and tuning, augmentation in training and inference, where they pool predictions over 22 variations of the image, as well as a higher resolution, a bigger model, pre-trained weights and external data.

We have made use of none of these techniques here because the focus of our experiments was not to achieve the highest classi cation performance on the ISIC dataset, but to compare the CBMIR performance of features extractors obtained from CNN classi ers and VAEs in the fairest way possible. For this objective, the most important precaution is that the architectures, procedures and data used in both approaches should be identical, and that was guaranteed in our experiments with the baseline CNN.

Hence, the purpose of this section and Table 3 is merely to ensure the supervised models had been well trained in the classi cation task by checking whether the performance is compatible with the literature.

The performance of .99 ROC-AUC on the CelebA dataset may raise suspicion, but it is

Table 3 – Classi cation performance of the supervised and mixed models.

	CelebA (Male)		ISIC (Melanoma)	
	Accuracy	ROC-AUC	Accuracy	ROC-AUC
Baseline CNN	0.981736	0.996912	0.852770	0.817767
SVAE	0.981174	0.994414	0.852041	0.814630
Ref.	0.98- - - -	-	0.882144	0.891850

Source: Research data. Reference scores come from the CelebA dataset paper (LIU et al., 2015) and from the best single-model entry on the leaderboard of the ISIC Challenge (2019)

compatible with the reference, and we ensured that the training and validation splits were disjoint with respect to the identity of the persons in the photos, and the validation dataset was not used for any hyperparameter tuning. This result means that the trained CNN is an appropriate and well-trained representative of the approach of CNN-based descriptors for CBIR, and that any advantage of the unsupervised models cannot be attributed to an improper training procedure for the CNN.

The classi cation performance on the ISIC dataset is lower, but it is also compatible with the reference. Also, we followed the same training procedures as ins the CelebA dataset, with the exception of the number of training epochs, which is limited by the size of the dataset, as explained in section 5.3.3.2. We attribute that difference to the ISIC dataset representing a more challenging task, less training images at 24k, and class imbalance (17% incidence of the Melanoma class).

5.4.2 Retrieval on CelebA Dataset

Figure 22(a) shows the mAP of each attribute for the three models. Under this metric, the VAE performed better than the CNN in only 11 out of 40 attributes, and the SVAE performed better than the CNN in 27 out of 40. At rst glance the retrieval performance of the VAE may appear disappointing when compared to the CNN. However, we urge the reader to see these results from a broader perspective, and take the conditions of the experiment into consideration, such as:

1. The VAE is a completely unsupervised model. Contrary to the CNN, it received no labels during training. The fact that the VAE's retrieval performance even came close to that of the CNN opens the possibility to learning high level descriptors sensitive to domain-speci c semantics, even without any structured annotation. This ability is extremely relevant in the context of medical images, were the cost of high quality, low noise, and structured annotation is high due to the expertise and quali cation required of the labeler. And it is even more relevant in the context of Computer-Aided Diagnosis, where the intended production environments are already lled with data with incomplete and unstructured class metadata.

Figure 22 – Retrieval performance on the CelebA dataset

Source: Elaborated by the author.

Description:(a) shows mAP, (b) shows P@10. Attributes on each graph are sorted by the score of the CNN at the respective metric. The choice for lines instead of bars is merely to improve visibility in such a dense chart. The attribute in bold is the training class.

2. Most of the attributes where the CNN performed better are related to the sex of the person. Some that stand out are: Male (sex attribute itself), Heavy_Makeup, Wearing_Lipstick, No_Beard, Weareing_Earings, Wearing_Necktie, and 5o_Clock_Shadow.

Despite considering the previous results to be excellent when taking into account the asymmetries between supervised and unsupervised techniques, the motivations presented at section 4.7 led us to expect that the VAE would display retrieval performance superior to the CNN, even though the former is completely unsupervised. Furthermore, despite the mAP values, the images retrieved by the VAE were consistently more similar to the query images than those returned by the CNN.

After closer inspection of the benchmark results, we could see that the PR-curves of the models with unsupervised criteria indicate higher peaks of precision at the top of the rankings, which is the most relevant part of the retrieval for CAD, and that such behavior is less pronounced on the CNN. A study and discussion for why this behavior happens in the VAEs is left for future work, but we proceed to quantify the retrieval performance at the top of the rankings. Figure 22 (b) presents the P@10 of the models in the same scheme as the previous one.

The P@10 reveals that, when looking at the rst retrieved images, even the completely

Figure 23 – Retrievals over the CelebA Validation split

Source: Elaborated by the author.

Description:Retrievals for random queries with CNN (a), SVAE (b) and VAE (c) respectively over CelebA Validation split. Each column is a retrieval, the top image is the queried one.

unsupervised model outperforms the CNN in most of the attributes and by a noticeable margin. Notice again that the attributes where the CNN displays higher advantage are strongly related to sex, such as Heavy_Makeup, Wearing_Lipstick, and 5o_Clock_Shadow. And the mixed model outperforms the CNN in all attributes but sex itself.

A visual inspection of the retrieved images corroborates this result. Figure 23 presents the retrievals obtained from the 3 models for random query images. While the CNN is consistent in retrieving faces of the same sex as the query image, many other factors of variance such as hair, pose, and skin and background color are inconsistent. The VAE on the other hand mixes the sexes, but is much more consistent at these other visual attributes; the SVAE, as expected, mixes the strengths of the two.

5.4.3 Retrieval on ISIC dataset

The retrieval benchmark over the ISIC dataset (Figure 24) re ected a consistent superior retrieval performance by the VAE and SVAE over the CNN, and this time even on the mAP metric as well. The single class where the CNN performed better was that which it was trained on. We believe that the different result between the datasets is due to the hidden classes on the ISIC-2019 dataset being less related with the supervised one than in the case of the CelebA

Figure 24 – Retrieval performance on the ISIC dataset

Source: Elaborated by the author.

Description:(a) shows mAP, (b) show P@10. Attributes on each graph are sorted by the score of the CNN at the respective metric. The attribute in bold is the training class.

dataset.

Again, a visual inspection of the retrieved images corroborates the results. We can see in Figure 25 that images retrieved by the VAE are visually more similar to the query image than those retrieved by the CNN, and the SVAE mixes this visual similarity with the CNN's higher precision over the training class.

5.5 Experiments Conclusions

The experiments showed that Variational Auto-Encoders (VAEs) represent an effective approach to train image descriptors for image retrieval while having a signi cant advantage over the more common approach of classi cation models: VAEs can be trained without annotation. This is extremely relevant for the medical domain were data with structured and standardized labeling is scarce, but unlabeled data is plentiful. Once we had the descriptors, those commonly large databases of images linked to patient history could be used with a CBMIR to aid diagnosis. But to train the descriptors with DL classi ers, a laborious and expensive labeling process would be required. VAEs, on the other hand, could be used to train DL descriptors on such datasets as they are.

This ability alone would make VAEs valuable tools for the elds of Content-Based Medical Image Retrieval and Computer-Aided Diagnosis, even if the quality of the retrievals obtained by classi es were better than those of VAEs. Nonetheless, our results show that VAEs can produce better retrievals than CNNs despite being unsupervised.

Upon a visual inspection, the images retrieved by our VAE are consistently more similar to the queried one than those retrieved by the baseline CNN, but retrieval metrics over the training class do not re ect this gap. The metrics over hidden classes, however, show that VAEs can

Figure 25 – Retrieval over the ISIC Validation split

Source: Elaborated by the author.

Description:Retrieval for random queries with CNN (a), SVAE (b) and VAE (c) respectively over ISIC Validation split. Each column is a retrieval, the top image is the queried one.

perform better at CBIR than CNNs, especially at the top of the retrieved ranking, and visual inspections corroborate this result.

However, it should be noted that, as expected, the baseline CNN performs better than the proposed models at the training class. Therefore, if there are labels available for the class of interest and only inter-class variations and class hit rate are important for the similarity criterion, then the classi cation approach used in the baseline CNN should be the better option.

But if there are no labels available, VAEs are a viable option to train domain speci c descriptors, and will even surpass the retrieval performance of the classi cation approach in overall visual similarity and intra-class variations. And nally, if both inter and intra-class variations are important and there are labels available, the proposed SVAE mixes the strengths of the two previous approaches with minimal performance trade-off. And since both inter-class and intra-class variations are important in CBMIR, we recommend employing the SVAE for scenarios where there are labels and the VAE when they are unavailable.