




Master in High Performance Computing

# Compressing Medical Images with Minimal Information Loss

*Supervisor(s)*:
Alessandro Laio,
Luca Heltai,
Alberto Sartori

*Candidate*:
Federico Barone

5th edition
2018–2019

# Acknowledgements

I would like to gratefully acknowledge CRO Aviano for making this project possible and the Abdus Salam International Centre for Theoretical Physics (ICTP) for funding my participation in the MHPC programme.

I sincerely thank my advisor Alessandro Laio for his guidance and dedication. Working with him and his research group has been an enriching experience. I also like to thank my co-supervisors Luca Heltai and Alberto Sartori for providing me the opportunity to do this project and for making sure everything went smoothly. I extend my gratitude to Ivan Girotto for his constant support and guidance all throughout this last year.

To all the students of the master, including also Nicolas and once again Alberto, I spent an excellent year working with you.

Lastly I would like to thank Clara for the company and for encouraging me to follow this new career path.

# Abstract

This thesis aims to explore the potentialities of neural networks as compression algorithms for medical images. The objective is to develop a compressed image representation suitable for image comparison. In particular we studied different autoencoder architectures, varying the encoding mechanism in order to achieve a high degree of compression while also retaining a meaningful feature space. Our work is focused on mammograms but the methods introduced here can be extrapolated to other types of medical images.

# Contents

# 1 Introduction

With the recent digitization of medical files, the amount of available medical data is growing tremendously. Exploiting this resource is a great opportunity and challenge for clinical and translational researchers. Big data and multi-omics approaches are believed to be the cornerstone of precision medicine [1]. At the moment, the biggest drawback for these approaches is the lack of standardized protocols capable of dealing with the information flow. Integrated platforms capable to aggregate and centralize the wide variety of data streams are crucial although difficult to develop.

In this context, the Oncology Research Center of Aviano (CRO Aviano) is developing the SENECA project (SEarch, iNtEgrate, extraCt and Analyze): a multi-parametric data analysis and management platform for clinical and translational research.

The present thesis is part of a collaboration between CRO Aviano and the Master of High Performance Computing (MHPC) within the framework of the SENECA project. The objective was to develop tools based on machine learning for improving medical image accessibility and analysis. In particular we built a compression algorithm which generates a standardized image representation suitable for future integration with different data sources, for example with clinical reports.

Due to data availability and relative lower complexity, we focused our work on breast cancer, particularly mammograms. This is the first step of the collaboration, which in the future aims to extend to different types of medical images, as could be Magnetic Resonance Imaging (MRI) or Computerized Tomography (CT) scans.

## 1.1 Background

Breast cancer is the leading cause of cancer related death in woman worldwide, representing approximately 15% of the total [2]. To date, X-ray mammography is the most frequent image testing method used to detect the disease. In particular the annual screening program for woman aged 50-69 has shown to be one of the most effective prevention measures, increasing early diagnoses cases and thus reducing mortality rates.

CRO Aviano has at least five years worth of data coming from mammogram screenings performed at their facilities. Furthermore each exam has its correspondent radiology report.

Aside from data availability, this last fact makes breast cancer mammograms an ideal test case; in the future we could explore building analysis tools around different data sources. In fact a concurrent project within the collaboration between CRO Aviano and the MHPC is based on standardizing the information contained in clinical reports.

## 1.2 Project overview

The present project is focused on building compression algorithms based on neural networks for mammogram images. The coding generated by such algorithm has to store the relevant information of the exam in a standardized way such that in the future it can be use with additional data coming from different sources (molecular assessments, radiology reports, etc.).

In Chapter 2 we described the main characteristics and format of our data together with the data preparation process.

Chapter 3 is divided in two. The first part is intended as a general introduction to compression algorithms and what differences traditional methods, in particular `JPEG`, from the one we developed. The second part introduces the flow chart of our algorithm, not going into the details of the compression mechanism but focusing on the overall pipeline the images undergo.

Chapter 4 introduces the concept of feature extraction in the framework of neural networks. This is the underlying process of our compression algorithm. It then goes on to describe the implementation details of the different versions

of the algorithm we developed: Convolutional Autoencoders and Asymmetric Autoencoders. The results obtained are discussed in Chapter 5.

Lastly, Chapter 6 is dedicated to a 2019 research paper written by Geras et al. [3] in which they implemented a deep neural network model to classify the presence or absence of cancer in full breast images. Although it is not strictly related to image compression, the paper was a constant reference throughout our work, as it gave a lot of insight on how to work with neural networks applied to mammograms. Moreover, we ported their model to the platform of CRO Aviano as it is a state of the art algorithm with multiple applications for clinical research and clinical practice.

# 2 Data

A routine mammogram exam consists of two standard views per breast, totalling four images: the craniocaudal (CC) or top view and the mediolateral oblique (MLO) or side view. Figure 2.1 shows an example of all four views.

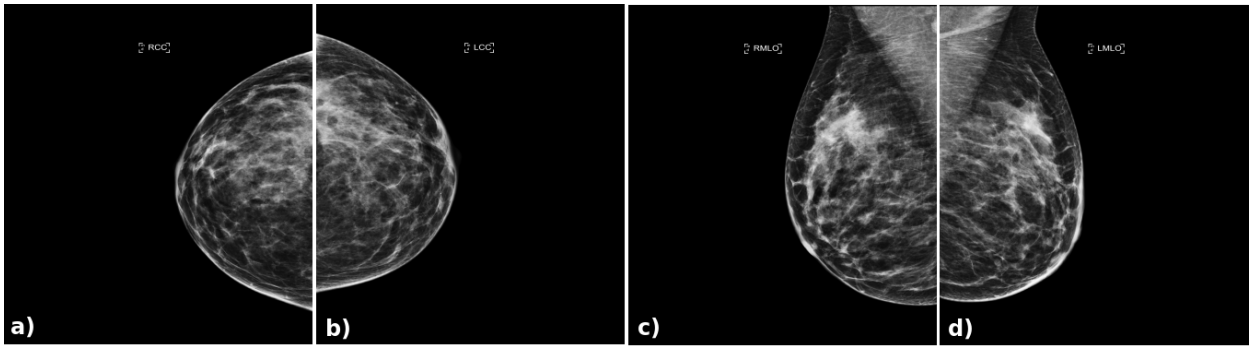


Figure 2.1: Example of a routine mammogram exam: a) right CC view; b) left CC view; c) right MLO view; d) left MLO view.

Mammograms are acquired on x-ray film, screen-film mammography (SFM), or more recently as full-field digital mammography (FFDM). The latter has better contrast resolution and is easier to store and transfer. Yet, due to budget restrictions, many hospitals still use SFM.

If digitized, each image is saved as a DICOM file, which stores the pixel array and its metadata. DICOM is a standard for handling, storing and sharing medical images. The file format can even handle the associated radiology report.

The report often contains the details of the procedure, the clinical history of the patient and the assessment of findings. The Breast Imaging-Reporting and Data System (BI-RADS) is a worldwide protocol used for writing these reports. It also defines assessment categories for the findings (Table 2.1).

| BI-RAD category | Description |
|---|---|
| 0 | Inconclusive. Additional studies are required. |
| 1 | Negative. |
| 2 | Benign. A definitive benign finding or findings. |
| 3 | Probably benign. A follow-up is recommended. |
| 4 | Suspicious abnormality. Biopsy is recommended. |
| 5 | Highly suggestive of malignancy. |
| 6 | Known cancer. Biopsy proven malignancy. |

Table 2.1: Breast Imaging-Reporting and Data System (BI-RADS) categories.

The BI-RADS standardized category system intends to establish an unambiguous way of labeling the reports. For our purpose they could be useful because they provide a ground truth. However, this ground truth is noisy, since they correspond to a confidence level of the radiologist which can later be proven right or wrong.

In this project we worked with two different datasets. The full image dataset (Section 2.1) provided by CRO Aviano used for the full breast classifier [3] and a patch-level dataset (Section 2.2) created from the previous one which we used to develop our compression algorithms.

## 2.1 Full image dataset

The full image dataset was provided by CRO Aviano. It consists of 1050 exams, each corresponding to a different anonymous patient, totalling 4000 images approximately (some exams have more or less than 4 images). The mammograms are FFDM stored as DICOM files.

Images have a standard resolution of $2560 \times 3328$ pixels, except for a minority with a higher resolution of $3328 \times 4096$ pixels.

The dataset was filtered to avoid corrupted mammograms. Data filtering was done using DICOM metadata information or applying image filters based on pixel intensity histograms. Figure 2.2 shows some rejected cases.

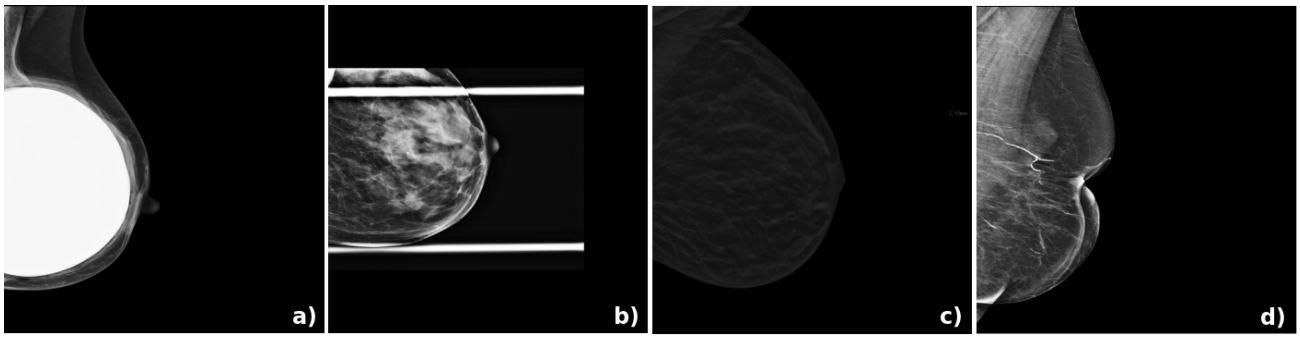


Figure 2.2: Examples of rejected images: a) breast with implant; b) image cropped and with additional markers ; c) image with modified contrast; d) breast after an intervention.

For example, Figure 2.2 a) and c) can be filtered checking the DICOM metadata, specifically the `Breast Implant Present` tag and when aluminum corresponds to the `Filter Material` tag. Figure 2.2 b) can be filtered using a threshold in a pixel intensity histogram. Figure 2.2 d) is a special case that we had to manually removed.

Exams were also filtered if they didn't have a radiology report with a valid BI-RADS label.

After data preparation, the full image dataset consisted of 662 complete exams including 2648 images. This means containing at least one image per view and the corresponding radiology report.

The full image dataset was use to evaluate the breast cancer classification model in Chapter 6.

## 2.2 Patch level dataset

The patch level dataset consists of 73400 patches of size 256×256 pixels extracted from 734 full size CC mammograms. Right CC views were flipped so that patches have all the same orientation. The original images are part of the CRO Aviano full image dataset. Mammograms where first filtered automatically as we described in the previous section, but a further manual selection was done to make sure we had the best samples to train our neural network. There are two main reasons why we needed a new dataset: the first is that

662 exams are not enough to train a neural network and the second is that the resolution of the full images is too big and would result in a neural network with too many parameters to train.

To generate the patches we created an ad hoc Python package, `mm_patch`, available online in the thesis repository [4]. The two main modules of the library are:

`crop.py` - Defines a set of cropping operations applied on the full image:

> `crop_img_from_largest_connected`: cropping function designed for mammograms and developed by N. Wu [5]. It crops out any intruding object not corresponding to the breast. For example the shoulders or belly sometimes appear in the mammography.
>
> `crop_horizontal/vertical`: Applied after the previous function, it further crops the image in order to extract patches avoiding the border of the breasts.

`extract.py` - Contains the extraction routine.

> extract_patches_2d: It is an enhanced version of `sklearn` function `.image.extract_patches_2d`. The patches are randomly selected from the full image, using a variable stride, adding flexibility in the selection of the patches: they can or cannot overlap depending of the stride parameter. An additional filter ensures no patch is extracted from the borders of the mammogram.

All the operations are handle by the class `data.Patches`, which takes as input the folder containing the full images and outputs the extracted patches with the desired characteristics: size, overlapping, filters, etc.

Figure 2.3 shows an example of the cropping and extraction mechanism. The light blue rectangle corresponds to the first cropping operation performed by `crop_img_from_largest_connected`. Note that it avoids the tissue from the bottom of the image as it is not part of the breast. The yellow rectangle corresponds to the additional vertical and horizontal cropping. From within the yellow region, the blue squares correspond to randomly extracted patches. Red squared patches are discarded because they contain the border of the breast.

We took a maximum of 100 patches per image, with a minimum horizontal and vertical stride of 50 pixels between patches.

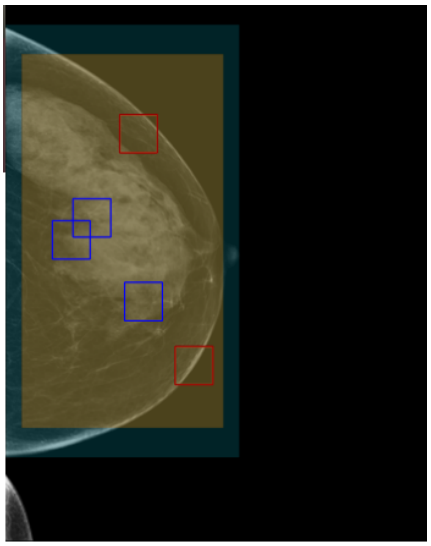

Figure 2.3: Visualization of the patch extraction algorithm.

# 3 Image Compression

Data compression refers to any process whose objective is to reduce the amount of bits needed to store information. Image compression is a subset of such algorithms aimed at compressing digital images. Figure 3.1 shows the general scheme of a compression algorithm. Note that it includes both compression and decompression mechanisms.

The input, $X$, is encoded into a compressed representation, $Z = E(X)$, which can then be decoded into a reconstructed version of the input, $X' = D(Z)$. Compression can be lossless or lossy, depending on the type of reconstruction.

**Lossless compression** or reversible compression guarantees no information loss in the encoding-decoding process. This means that the reconstructed input is equal to the original input, $X' = X$. The compression is achieved by eliminating redundant information and choosing an appropriate encoding depending on the characteristics of the target data.

**Lossy compression** or irreversible compression identifies marginal information and discards it in order to obtain a smaller compressed representation. As a consequence, the reconstructed input is a distorted version of the input, $X' \simeq X$.

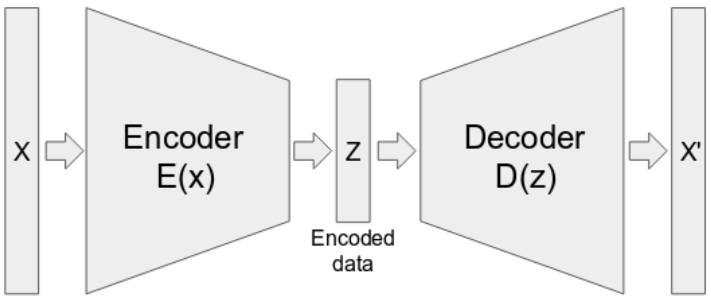


Figure 3.1: Compression process.

Which information should be kept and which is discarded depends on the target data and the specific application.

Lossless algorithms are usually used for text base data, where information loss is not acceptable. On the other hand, lossy algorithms are used for audio or images where a degree of distortion is acceptable. There is always a trade off between the level of compression and the quality of the reconstruction. What is considered acceptable depends on the application.

For images, the standard compressed formats are `PNG` or `JPEG 2000`. The former is lossless while the latter is lossy. Both algorithms are extremely efficient and would work just fine with our type of images. The follow-up question is: Why didn't we use them instead of developing our own algorithm? The main reason is that we were seeking for a compressor that generated a meaningful encoding. More specifically a set of numbers or vector, in which each coordinate corresponds to a feature of the image. In this way, images are mapped into a vector space were distances corresponds to similarities, not only pixel to pixel, but also feature-wise. The second reason is that an algorithm tailored to a specific dataset can potentially outperform a general purpose one, like `PNG` or `JPEG 2000`.

To sum up, we were looking for a compressor that generated a meaningful coding that was well suited for data analysis and data integration while achieving the maximum level of compression possible. Starting from next section, the thesis is dedicated to describe in detail the mammogram compression algorithm we developed together with its performance.

## 3.1 Mammogram compression

Figure 3.2 shows the work flow of the mammogram compression algorithm we developed. It is a lossy compressor based on neural networks, trained specifically on our dataset. Because of the size of the images, the actual encoder-decoder works at a patch level, meaning we divide the input into smaller blocks of fix size which are compressed independently and stored as a single coding. Image reconstruction is done by decoding each block and arranging them according to their original position.

The entire pipeline is as follows:

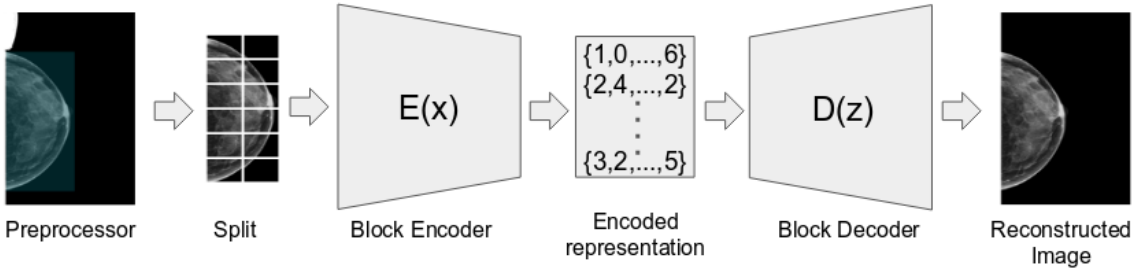


Figure 3.2: Schematic diagram of the mammogram compression algorithm.

1. Preprocessor: The input image is cropped to keep only the relevant breast regions. In this way we discard most of the background and any region not corresponding to the breast. The original image size is stored as metadata. The function used, `crop_img_from_largest_connected`, was already introduced in Section 2.2.
2. Split: The now cropped image is divided into non-overlapping blocks of size 256×256. The number of blocks per row and column are also stored as metadata. If needed, the cropped image is padded so that its width and height are multiple of 256.
3. Encode: The blocks are fed one by one into the encoder neural network. Each block is mapped into a vector of fix size. The coding is concatenation of all the compressed vectors stored in row major order.
4. Decode: Each vector is fed into the decoding neural network to obtain the reconstructed patches.
5. Reassemble: Knowing the number of blocks per row and columns, the full image is reassemble. If needed, background is added to match the original image size.

The compressed image is stored as a binary file containing the concatenated codings of the blocks that compose the full image. The level of compression achieved can be quantify using the compression factor (CF):

$$\mathrm{CF} = \frac{\text{Input size}}{\text{Coding size}}. \tag{3.1}$$

CF depends strictly on the characteristics of the encoding-decoding neural

networks. The next chapter is dedicated entirely on this subject.

# 4 Feature Extraction

With the objective of building a compression algorithm that produces a meaningful coding, we explored feature extraction techniques based on neural networks as encoding mechanisms.

In machine learning, dimensionality reduction refers to any process which aims at reducing the number of variables needed to describe a particular dataset. Feature extraction is a subset of these techniques, in which dimensional reduction is achieved by combining the original variables into a smaller but still representative set of features.

Feature extraction algorithms are labeled as linear or non linear depending on the underlying transformation process. Neural networks fall on the latter as they imply a highly non linear transformation of the input optimized to produce a specific output.

The benefits of this non linear techniques can be intuitively understood based on the assumption that the important content of a high dimensional dataset belongs to a manifold with intrinsic dimension much lower than the original number of coordinates (see Figure 4.1 for a basic example). Finding the parametric representation of the manifold would mean finding its minimal representation.

Ideally, this representation would include all the information content of the original dataset. In practice, the content of the feature representation depends on the extraction algorithm. Moreover, the relevance of the information is determined by the subsequent task one wants to perform. In the case of neural networks this can be shaped depending on the loss/objective function used for optimizing the network.

To sum up, feature extraction algorithms based on neural networks can be tailored to build a meaningful and compressed representation of a specific dataset. This makes them a perfect candidate for our compression algorithm.

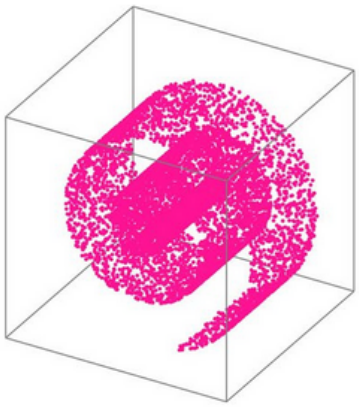

Figure 4.1: 2d Swiss Roll embedded in a 3d space. In this example the data lay on a surface that can be parametrized using two coordinates thus its intrinsic dimension is 2 while the embedding dimension is 3.

For this project we worked with two types of neural networks as tools of feature extraction: **autoencoders** and **classifiers**. Their main difference is the way in which they learn; respectively in an **unsupervised** or **supervised** way. This has a direct consequence in the representations they extract.

Autoencoders guarantees proper image reconstruction while classifiers can produce a feature representation that captures image semantics as per human perception.

What follows is a detailed description of the specific architectures we implemented together with the notions that support our choice.

## 4.1 Autoencoders

Autoencoders [6] are the natural choice when attempting image compression with neural networks. They consist of an encoder-decoder pair optimized to output a reconstructed version of the input.

Internally this type of networks are composed by a series of convolutional or fully connected layers, with the dimension of the input matching the dimension of the output. In between the input and output there is a bottleneck which

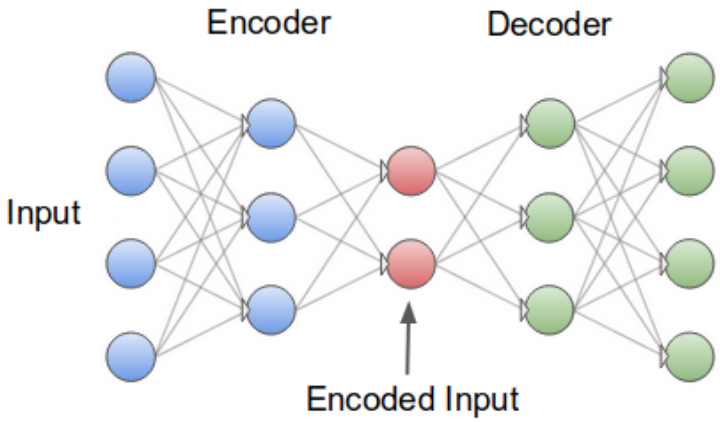


Figure 4.2: Autoencoder schematic.

enforces a dimensional reduction of the input data. This layer divides the encoder from the decoder. The activation of the bottleneck layer is known as the *latent space* or *coding*. Figure 4.2 clarifies this description.

Encoder and decoder have usually a mirror architecture, as ideally the latter has to perform the inverse transformation of the former.

Autoencoders are trained in an **unsupervised** manner by minimizing the reconstruction loss of the data, generally written as the mean squared error (MSE):

$$L(\omega) = \sum_i \|X_i - X_i'\|^2 = \sum_i \|X_i - D(E(X_i))\|^2, \tag{4.1}$$

where $E$ and $D$ stands for the non linear mapping of the encoder and decoder respectively, $X$ is the input, $X'$ the output and $\omega$ the parameters of the network. The sum runs over a set of data levelled by $i$.

The features learned by an autoencoder on a set of images are such that one can fully reconstruct the original images from it. This means that, for example, the coding should retain the spatial location of the 'objects' in the image.

Tweaking the loss function changes the nature of the features, enabling the coding to capture abstract image semantics, but also changing the network output.

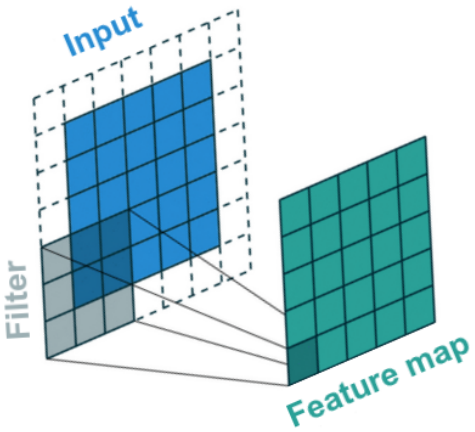


Figure 4.3: Example of a convolution operation with padding to mantain the same input-ouput size. Modified version of figure taken from Medium's convolutional layers guide

Lots of work have been done in this direction, developing tailored architectures known as Denoising Autoencoders (DAE) [7], Generalized Autoencoders (GAE) [8] or Relational Autoencoders (RAE) [9].

In our case we used MSE loss function and its regularized variants [10] because faithful reconstruction of the input is required.

First we implemented a vanilla **Convolutional Autoencoder (CAE)** which we later enhanced by adding two fully connected layers as a Principal Component Analysis (PCA) proxy to further decrease the dimension of the latent space. We called this implementation **Convolutional Autoencoder plus PCA**.

### 4.1.1 Convolutional Autoencoder

Convolutional Autoencoders (CAE) are regular autoencoders composed by a series of convolutional layers.

Figure 4.3 shows a convolution operation. The **kernel** or **filter** slides along the input and generates an output by sequentially multiplying and adding the parameters of the kernel with the corresponding slice of image. The output is called **feature map** or convolved feature.

A convolutional layer is composed by a fixed amount of independent filters, all of the same size. Each of them generates a 2d feature map. The layer output is a cube consisting of the stacked features. The number of maps, also known as **channels**, corresponds to the depth of the cube.

Convolutional layers are commonly used when dealing with images because of their shared weight architecture which considerably reduces the amount of parameters in the network.

Note that if the input has more than one channel, the 2d filter of Figure 4.3 becomes a 3d filter of depth equal to the number of input channels.

### Network implementation

Defining the autoencoder architecture for our specific problem was a trial and error process. The extensive literature available served as a general guideline although there were not many examples that dealt with an input image of size 256×256 pixels [11][12]. Typically examples were done using MNIST (28x28 pixels) or CIFAR (32x32 pixels) [13].

Figure 4.4 shows the scheme of our CAE. Encoder and decoder have a mirror architecture, each containing 5 layers. The volumes in the figure represent the output of each convolutional layer.

All layers are composed by the following components:

- **Convolutional layer** with kernel size 3x3 and padding 1x1. The padding extends with zeros the input channels, which is the default behaviour in PyTorch. In this configuration the convolutional layer changes only the number of channels while maintaining the size of the feature maps fixed.
- **ReLU activation** function except for the output layer for which we used a sigmoid function.
- **Pooling/Unpooling layer** which downsamples/upsamples each channel depending if in the encoder or decoder. Downsampling was performed by a max pooling operation with kernel size 2x2. Upsampling was performed with the 'nearest' method with a kernel of size 2x2. The last layer doesn't include a pooling operation.

The configuration is such that in each layer we increased (decreased) the number of channels by 2 while decreasing (increasing) the size of the feature maps by

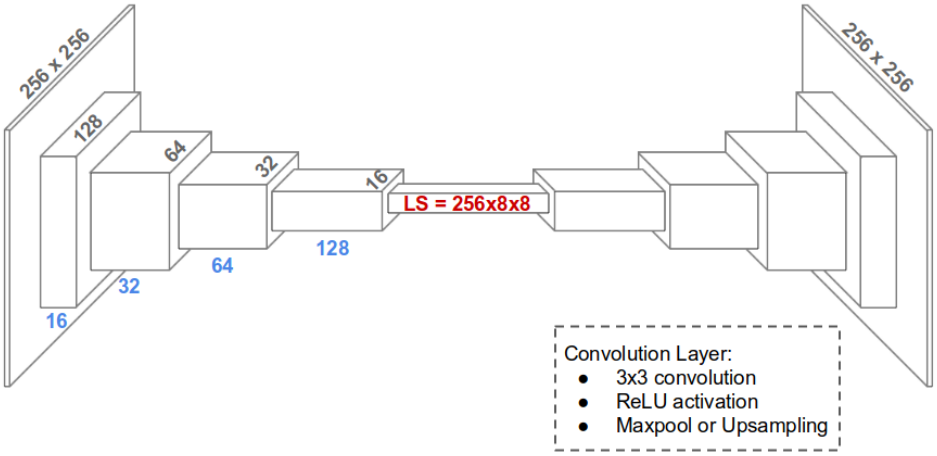


Figure 4.4: Convolutional autoencoder schematic. Light blue numbers indicate the number of channels in each layer. Black numbers indicate one size of the squared future maps. In red the size of the latent space.

the same factor. This is an heuristic rule which is commonly used when dealing with Convolutional Neural Networks. The details of the network are summed up in Table 4.1.

As previously mentioned, the input image was of size 256×256 pixels. The smaller latent space achieved was of size 256×8×8 (256 channels of size 8×8 each). In this case we say we reached a **compression factor** of 4, meaning that the coding is four times smaller than the original image.

Further attempts to compress the input using this type of layers resulted in poor image reconstruction. We also implemented alternative versions of this architecture: adding batch normalization in each layer, using convolutional and transpose convolutional layers without padding instead of pooling layers or smoothing the downsampling/upsampling rate by making the network deeper. Results were at best equal than before.

Although we didn't exhaust all the possibilities using this type of convolutional networks, we managed to increase the compression factor by at least an order of magnitude. The following section explains in detail this implementation.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 16, 256, 256] | 160 |
| MaxPool2d-2 | [-1, 16, 128, 128] | 0 |
| Conv2d-3 | [-1, 32, 128, 128] | 4,640 |
| MaxPool2d-4 | [-1, 32, 64, 64] | 0 |
| Conv2d-5 | [-1, 64, 64, 64] | 18,496 |
| MaxPool2d-6 | [-1, 64, 32, 32] | 0 |
| Conv2d-7 | [-1, 128, 32, 32] | 73,856 |
| MaxPool2d-8 | [-1, 128, 16, 16] | 0 |
| Conv2d-9 | [-1, 256, 16, 16] | 295,168 |
| MaxPool2d-10 | [-1, 256, 8, 8] | 0 |
| Conv2d-11 | [-1, 128, 8, 8] | 295,040 |
| Upsample-12 | [-1, 128, 16, 16] | 0 |
| Conv2d-13 | [-1, 64, 16, 16] | 73,792 |
| Upsample-14 | [-1, 64, 32, 32] | 0 |
| Conv2d-15 | [-1, 32, 32, 32] | 18,464 |
| Upsample-16 | [-1, 32, 64, 64] | 0 |
| Conv2d-17 | [-1, 16, 64, 64] | 4,624 |
| Upsample-18 | [-1, 16, 128, 128] | 0 |
| Conv2d-19 | [-1, 1, 128, 128] | 145 |
| Upsample-20 | [-1, 1, 256, 256] | 0 |

Table 4.1: Architecture layout for the Convolutional Autoencoder. It has 784,385 trainable parameters. The colored row corresponds to the output of the encoder. The indexes of the output shape are [batch size, channels, x, y].

### 4.1.2 Convolutional Autoencoder + PCA

Our CAE implementation achieved a compression factor equal to 4 (see Section 4.1.1). To further compress the latent space we added two fully connected layers. This layers acts as a proxy of a PCA transformation performed on the latent space of the already trained CAE. We called the resulting network CAE+PCA.

This section is divided in three. First we will give an overview of PCA. Secondly we will describe its relationship with neural networks. Lastly we will give the details of how we trained and implemented the CAE+PCA network.

## PCA

Principal component analysis (PCA) is an orthogonal transformation of a dataset into a set of linearly uncorrelated variables [14]. The axis of the new coordinate system corresponds to the directions, principal components, along which the variation in the data is maximal. The coordinates are ranked such that the first component has the greatest variance, the second component has the second greatest variance and so on.

To define the PCA transformation we need to find the principal components. Lets start by the first component.

In mathematical terms, we want to find the direction which maximizes the variance of a given dataset $X$. For simplicity we will assume $X$ is centered, namely the average of $X$ is zero. The variance on a given direction can be expressed as follows:

$$\mathrm{Var}_w(X) = w^T X^T X w, \tag{4.2}$$

where $w$ is the unitary vector representing the direction. Note that $X^T X$ is the covariance matrix, $C$, of the dataset.

Reformulating the problem, we want to find

$$w_1 = \underset{\|w\|=1}{\mathrm{argmax}} \{w^T C w\}. \tag{4.3}$$

By construction $C$ is a positive semidefinite real matrix, thus it has an orthonormal basis of eigenvectors. It is easy to see that Equation 4.3 is maximized when $w$ is equal to the eigenvector of $C$ associated with the biggest eigenvalue.

This result is also valid for the remaining principal components. This is,

$$w_k = v_k, \tag{4.4}$$

where $w_k$ is the $k$th principal component and $v_k$ is the kth eigenvector of $C$ sorted on the magnitude of the associated eigenvalue $\lambda_k$.

Once the principal components are known, the PCA transformation is defined as follows:

$$X_{new} = V^\top X, \tag{4.5}$$

where $V^\top$ is the linear transformation matrix which rows are the sorted eigenvectors of $C$: $\{v_1, v_2, ...\}$.

If used as a **linear dimensional reduction** mechanism, matrix $V^\top$ can be truncated in the number of rows, eliminating the eigenvectors which account for the smaller amount of variation of the data. These are the eigenvectors associated with the smaller eigenvalues. PCA then becomes an orthogonal projection onto the directions which explain the greatest part of the data variance.

### Single layered autoencoders and PCA

In 1989, Baldi and Hornik [15] studied the relationship between PCA and autoencoders. More precisely they showed that an autoencoder composed by a single linear layer, optimized on a MSE loss function (Eq. 4.1), has a unique global minimum corresponding to an orthogonal projection onto the subspace spanned by the first principal directions of a PCA transformation.

Although equivalent, the computational cost of training a linear autoencoder is much higher than performing a PCA reduction. The former is done by stochastic gradient descent (SGD) while the latter is a matrix decomposition algorithm.

If wanting to implement such autoencoder, an alternative from training it would be to use the projection values obtained by the corresponding PCA transformation. The procedure would be the following.

Figure 4.5 shows an example of a single layered autoencoder with no activation function. Given a data point $x \in \mathbb{R}^n$, the encoder performs a linear projection onto an m-dimensional subspace, $z \in \mathbb{R}^m$, defined as

$$z = \omega x + \beta, \tag{4.6}$$

where $\omega \in \mathbb{R}^{m \times n}$ are the weights of the encoder and $\beta \in \mathbb{R}^{m \times 1}$ its bias.

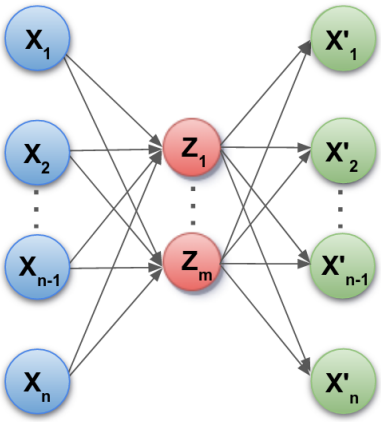


Figure 4.5: Schematic of a single linear layered autoencoder.

As described in Equation 4.5, the PCA equivalent projection would be,

$$z = V_m^\top (x - \bar{x}), \tag{4.7}$$

where $V_m^\top \in \mathbb{R}^{m \times n}$ is the truncated projection matrix including the first $m$ covariance eigenvectors and $\bar{x}$ is the mean value of the input dataset.

Equating this expression with the encoders equation (Eq. 4.6) we arrived to the first identity,

$$\omega = V_m^\top, \quad \beta = V_m^\top \bar{x}, \tag{4.8}$$

which allows identifying the parameters of the encoder.

For the decoder we have the following transformation,

$$x' = \omega' z + \beta' = \omega' (V_m^\top x - V_m^\top \bar{x}) + \beta'. \tag{4.9}$$

By requiring image reconstruction, namely $x \simeq x'$, we obtain our second identitiy,

$$\omega' = V_m, \quad \beta' = \bar{x}, \tag{4.10}$$

which allows identifying the parameters of the decoder.

To be rigorous, $x'$ is not equal to $x$ since there was loss of information when projecting the data onto the m-dimensional eigenvector subspace.

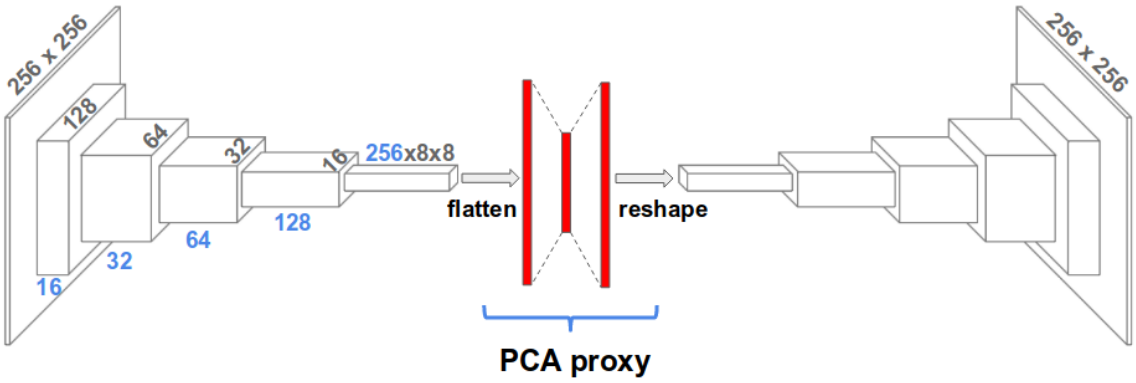


Figure 4.6: CAE+PCA schematic. Light blue numbers indicate the number of channels in each layer. Black numbers indicate one size of the squared future maps. In red the flattened CAE latent space plus the output of the two linear layers that compose the PCA proxy.

To summarize, using equations 4.8 and 4.10 we can implement an already trained single layered autoencoder with linear activation function. The computational cost is equal to the matrix decomposition used to perform PCA.

### Network implementation

Figure 4.6 shows the scheme of the CAE+PCA network. The architecture is the same as the CAE network (see Section 4.1.1), the only difference being the flattened CAE latent space plus two linear layers. Their purpose is to act as a PCA proxy aimed at further compressing the lattent space.

The configuration of the proxy is equal to the one layered autoencoder we described in the previous section (see Figure 4.5), yet the behavior is not exactly the same.

Being surrounded by convolutional layers and not trained directly to reconstruct its input, the two central layers perform a linear projection which is not necessary onto the subspace spanned by the principal directions of PCA.

Instead, the idea is to use the PCA parameters as initial weights for the linear layers. The training will then tweak their values to better fit the data. The correspondence between the initial weights and the parameters are given

by equations 4.8 and 4.10. In the following, the implementation steps are described:

1. Run the dataset you want to compress threw the pretrained CAE network and store its compressed representation. Note that the CAE network has to be trained on the same dataset.
2. Run PCA on the latent space and store its parameters (its principal directions). For this we used the python library `Scikit-learn`.
3. Load the CAE pretrained weights and the PCA parameters (Eq. 4.8 and 4.10) onto the CAE+PCA network.
4. Train the CAE+PCA network.

From Section 4.1.1 we have that for input images of size $256 \times 256$, the minimum coding size achieved with CAE was of size $256 \times 8 \times 8$.

Using this network as a base we implemented two CAE+PCA versions: one with a coding of size 1024 and another one with a coding of size 100. In Chapter 5 we analyze their performance.

## 4.2 Convolutional Neural Network classifiers

Ever since AlexNet [16] won the ILSVRC-2012 competition, Convolutional Neural Networks (CNNs) became the dominating algorithm in image classification problems.

Figure 4.7 shows the general scheme of a CNN classifier. The network is usually described as having two distinct modules, each with a different functionality:

- **feature extraction:** the part of the network in charge of feature learning. It is composed by a set of layers, each a composite function of a convolutional layer and a combination of batch normalization, ReLU activations or its variants, poolings layers, etc.
- **classifier:** a set of fully connected layers in charge of classifying the feature representation space according to the desired labels.

The division into modules is more conceptual than mathematical. It is based on past research [17][18] where they successfully used the feature extraction layers trained on a task and reused them on a second task.

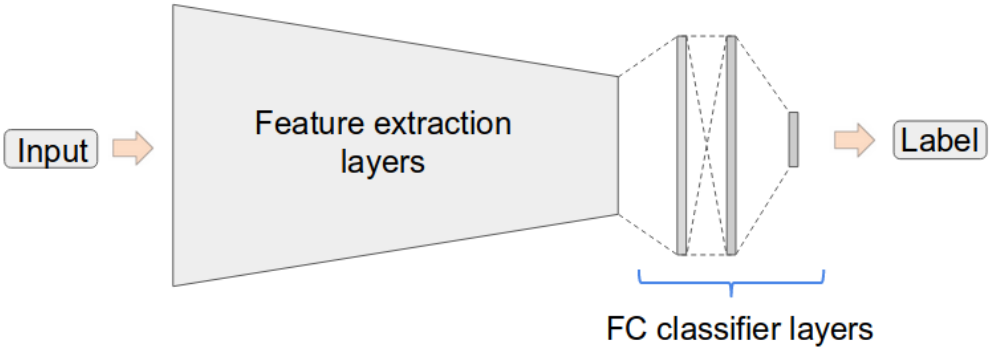


Figure 4.7: General architecture for a CNN classifier.

The current understanding is that the first convolutional layers learns generic features from the image, like edges and blobs. As you get deeper into the layers the features become more abstract and task related. The number of layers that can be reused depends on the problem.

This technique is called **transfer learning** [19] and it is one of the reasons CNNs became so popular.

Our interest in this method is because we want to reuse the feature extraction mechanism of a classifier network as the encoding of a compression algorithm.

In particular we will work with a CNN trained to detect the presence of malignant or benign cancer lesions on patches extracted from full mammographies. The network architecture is DenseNet-121 and it was trained and implemented by KJ. Geras et al. [3].

The advantage of this representation is that, being trained on cancer recognition, it is tailored to include meaningful features. The simplest example one can think of is that the information of a cancerous lesion is guaranteed to be captured in the coding. The objective is to compare this type of features with the ones learned by a traditional autoencoder (see Section 4.1).

The disadvantage of the technique is that it is a **supervised learning method**, meaning that it requires the labeled data to trained the classification algorithm. In our case this was not a problem since the trained model was available. If data is not available, unsupervised learning methods are the only alternative.

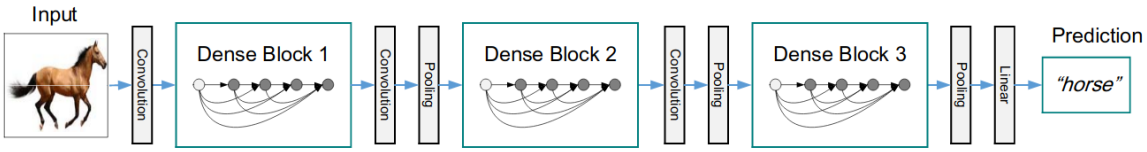


Figure 4.8: A DenseNet with three dense blocks. In between the blocks you have the transition layers. [20]

What follows is a detailed description of the Densenet architecture (Section 4.2.1) and how we incorporate its feature extraction layers into a compression algorithm which we called Asymmetric Autoencoder (Section 4.2.2).

### 4.2.1 DenseNet classifier

Densely connected Convolutional Neural Networks, or DenseNets [20], are one of the state of the art networks used for image classification. They emerged as a solution for the main problems faced by deep learning models: vanishing-gradient problem and the ever increasing number of parameters.

DenseNet addresses these issues by adding feed-forward connections between layers, so that each layer has direct access to its preceding feature maps. This reduces the loss of information along the network and gives each layer a direct access to the gradients from the loss function. Counterintuitively the amount of parameters needed is less if compared with CNNs of the same depth. To understand this last statement we need to look into the architecture of the network.

As shown in Figure 4.8, DenseNet is divided into dense blocks. Every block includes a sequence of dense layers, each a composite function of three consecutive operations: batch normalization, followed by a ReLU activation and a $3\times3$ convolution.

Inside each block the dimensions of the feature maps remains constant while the number of channels increases at a fix rate. The **growth rate** is a hyperparameter of the network and it is the same for all blocks. This means that each layer, independent of the block, learns $k$ new feature maps, where $k$ is the growth rate.

Each layer $l^{th}$ inside a block can be represented as,

$$x_l = H([x_0, x_1, ..., x_{l-1}]), \tag{4.11}$$

where $H$ is the layer non linear transformation and [$x_0$,$x_1$,...,$x_{l-1}$] refers to the concatenation of the feature-maps produced in layers 0,..., $l-1$.

The constant growth rate is the solution to over parametrization. As opposed to regular CNN were the number of output channels increases layer per layer, in DenseNet each layer added has to learn only $k$ new feature maps, where $k$ is usually small (order 10).

The missing piece in the description are the transition layers between dense blocks. These layers consist of a batch normalization layer and an $1\times1$ convolutional layer followed by a $2\times2$ average pooling layer with a stride of 2. They are in charge of reducing by half the size of the feature maps between blocks, thus reducing the overall amount of parameters of the network.

For further details about this architecture and its variants refer to the original paper [20] or to P. Ruiz's guide [21].

### DenseNet-121 implementation on cancer recognition

The deep DenseNet-121 we used for feature learning is an auxiliary network for the full breast DNN classifier developed by KJ. Geras et al. [3] and described in Chapter 6.

The network is a patch-level classifier where the labels are pixel level annotations of cancerous lesions. The classification task is detecting the presence of findings on a patch, either benign or malignant. The classified patches are later used to construct a heatmap of the full image which is used as an additional input for the breast-level model. With this approach the authors are trying to exploit precise local information (patch classification), together with global information (full breast classification).

DenseNet-121 architecture is composed of four dense blocks with 6, 12, 24, 16 dense layers respectively, followed by a linear layer which outputs the classification. In this version each layer includes an additional $1\times1$ convolution to further reduce the parameter space.

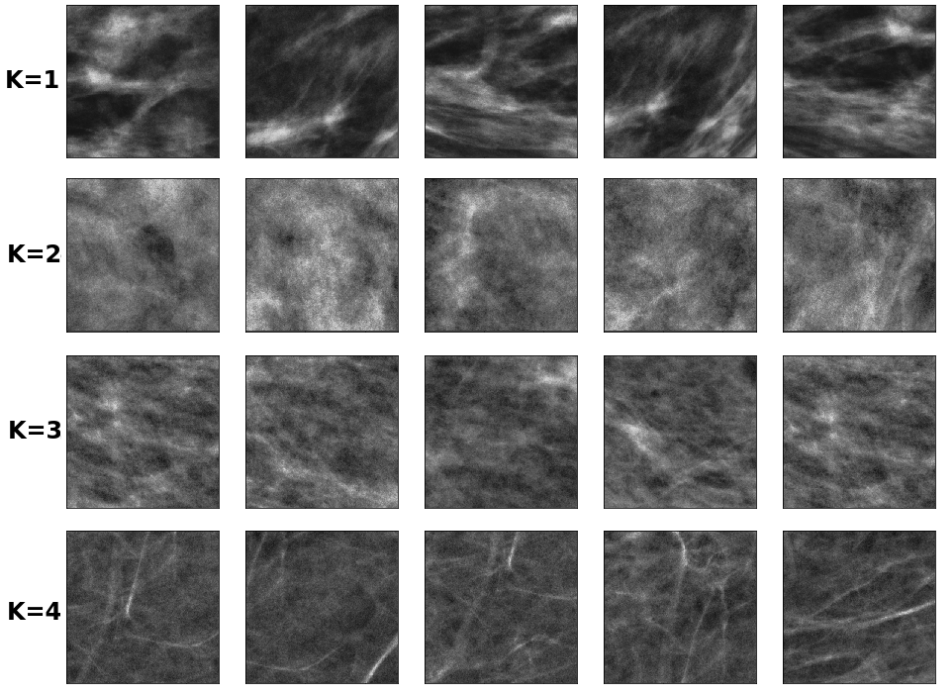


Figure 4.9: The random samples in each row correspond to a particular cluster found on the classifier's latent space.

To understand the image representations learned by the classifier we performed cluster analysis on the latent space generated when running the model on the patch-level dataset 2.2. In particular we ran a k-mean clustering algorithm implemented in `Scikit-learn`. The objective was to linearly split the space into $k = 10$ clusters and inspect if elements within each cluster shared visual features.

Figure 4.9 shows random samples extracted from different clusters, organised by row. Each cluster contains visually similar images although a pixel to pixel metric would not agree. This means that the euclidean metric, or any other variant, of the latent space generated by the classifier enables image comparison as per human perception.

We did the same study with the image representations of the different versions of the convolutional autoencoders but we didn't obtain the same results. This suggests that the clustering effect we observed is related to the cancer

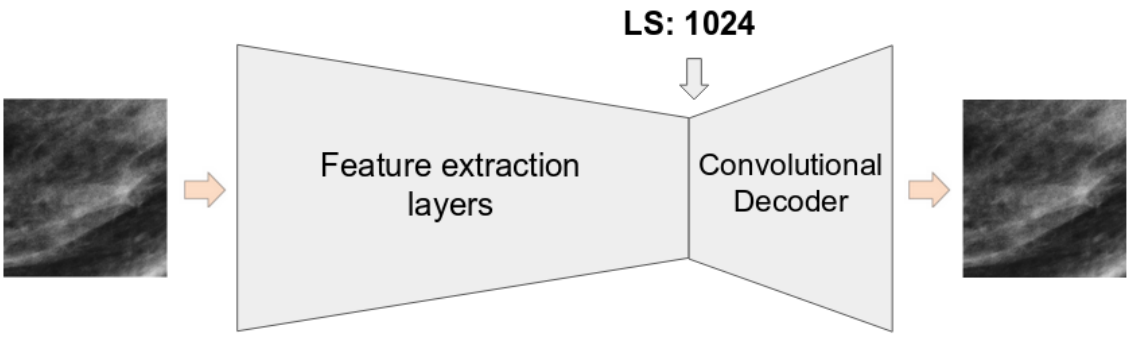


Figure 4.10: Asymmetric Autoencoder layout. When using DenseNet-121 feature extraction layers the latent space is of size 1024.

recognition task the classifier was trained to perform.

### 4.2.2 Asymmetric Autoencoder

In the previous section we described a DenseNet-121 trained on cancer recognition at a patch level. Our goal is to construct an autoencoder based on it.

To build the encoder we removed the classification layers from DenseNet-121, and used all four dense blocks as our feature extracting layers. The coding size, corresponding to the output of the last block, is of size 1024. Being the input image of resolution 256×256 pixels, this means a compression factor of 64.

What is left is to build the corresponding decoder. Ideally it should mirror the encoder architecture yet given the size of the encoder, 121 convolutional layers, and the lack of previous examples in the literature we opted to build a decoder with less layers and avoiding any type of skip connections as the ones present in DenseNet. Hence the name Asymmetric Autoencoder.

#### Network implementation

Figure 4.10 shows the block scheme of the network. Note that it is the same as a classifier, Figure 4.8, with the classification layers replaced by a decoder.

To build the decoder we tried several alternatives. We ended up using a 5 layered decoder, each a composite function of four consecutive operations: 3×3 convolution with padding, batch normalization, ReLU activation and 2×2 nearest upsampling.

The details of the networks are summed up in Table 4.2.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| NYU-DenseNet121 | [-1, 1024] | 6,953,856 |
| ConvTranspose2d-488 | [-1, 512, 4, 4] | 8,389,120 |
| ReLU-489 | [-1, 512, 4, 4] | 0 |
| Conv2d-490 | [-1, 256, 4, 4] | 1,179,904 |
| BatchNorm2d-491 | [-1, 256, 4, 4] | 512 |
| ReLU-492 | [-1, 256, 4, 4] | 0 |
| Upsample-493 | [-1, 256, 8, 8] | 0 |
| Conv2d-494 | [-1, 128, 8, 8] | 295,040 |
| BatchNorm2d-495 | [-1, 128, 8, 8] | 256 |
| ReLU-496 | [-1, 128, 8, 8] | 0 |
| Upsample-497 | [-1, 128, 16, 16] | 0 |
| Conv2d-498 | [-1, 64, 16, 16] | 73,792 |
| BatchNorm2d-499 | [-1, 64, 16, 16] | 128 |
| ReLU-500 | [-1, 64, 16, 16] | 0 |
| Upsample-501 | [-1, 64, 32, 32] | 0 |
| Conv2d-502 | [-1, 32, 32, 32] | 18,464 |
| BatchNorm2d-503 | [-1, 32, 32, 32] | 64 |
| ReLU-504 | [-1, 32, 32, 32] | 0 |
| Upsample-505 | [-1, 32, 64, 64] | 0 |
| Conv2d-506 | [-1, 16, 64, 64] | 4,624 |
| BatchNorm2d-507 | [-1, 16, 64, 64] | 32 |
| ReLU-508 | [-1, 16, 64, 64] | 0 |
| Upsample-509 | [-1, 16, 128, 128] | 0 |
| Conv2d-510 | [-1, 8, 128, 128] | 1,160 |
| BatchNorm2d-511 | [-1, 8, 128, 128] | 16 |
| ReLU-512 | [-1, 8, 128, 128] | 0 |
| Upsample-513 | [-1, 8, 256, 256] | 0 |
| Conv2d-514 | [-1, 1, 256, 256] | 73 |
| Generator256-515 | [-1, 1, 256, 256] | 0 |

Table 4.2: Architecture layout for the decoder of the Asymmetric Autoencoder. It has 9,963,185 trainable parameters. The colored row corresponds to the output of the encoder. The indexes of the output shape are [batch size, channels, x, y].

# 5 Training and results

This chapter discusses the results obtained by the three different versions of the mammogram compression algorithm we developed. Each version is based on one of the following networks, all trained on the patch-level dataset (Section 2.2):

- Convolutional Autoencoder
- Convolutional Autoencoder + PCA
- Asymmetric Autoencoder

All networks were implemented using the open source Python library `Pytorch`. Some details of the training will refer to specific functions from that library. Equivalent implementations can easily be found in any other machine learning framework.

The hardware used was a GeForce GTX TITAN X with 12 GB of VRAM and Maxwell architecture.

The models, its trained weights and the training routines are all available online in the repository of the project [4].

## 5.1 Convolutional Autoencoder

The architecture of the Convolution Autoencoder (CAE) we implemented was described in Section 4.1.1. It is a 10 layered encoder-decoder with a latent space of size $256\times8\times8$. The size of the bottleneck was decided empirically; it is a trade off between the degree of compression and the quality of the reconstructed image.

To measure the degree of compression we used the compression factor (eq. 3.1) that for this particular network was:

$$\mathrm{CF} = \frac{256 \times 256}{256 \times 8 \times 8} = 4. \tag{5.1}$$

To measure the quality of reconstruction we need to define a metric that represents the similarity between the input and the output image. A straight forward choice was using the Mean Square Error (MSE), as defined in equation 4.1. It is a pixel to pixel comparison between two images.

## Training

The training of the network is an optimization process that aims to minimize an objective or loss function. By defining the MSE as the loss function we are training the network to output a pixel to pixel reconstruction of the input. Moreover, we are jointly optimizing the image representation conditioned on this metric of similarity. This is what we meant previously by stating that the features learned depend on the targeted task.

The training configuration was the following:

```
samples = 10000
validation set = 20% of total
number of epochs = 300
batch size = 80
loss = Mean Squared Error
optimizer = Adam
learning rate = 0.0005
scheduler = ReduceLROnPlateau(patience = 8, factor = 0.5)
```

We trained the models with 10000 samples to speed up the training process. This was extremely useful while deciding the architecture. The performance didn't change when using the whole dataset, probably because 10000 patches was already a representative sample.

The optimizer that showed better performance was Adam optimizer in combination with the `ReduceLROnPlateau()` scheduler. The name refers to the

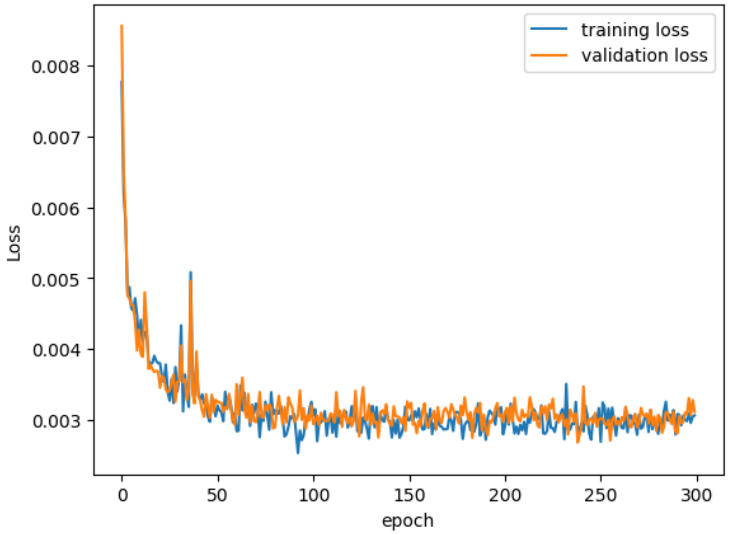


Figure 5.1: CAE training history. The loss function used was MSE.

`PyTorch` scheduler class: every 8 epochs, `patience`, it reduces the learning rate by a factor of 2 if there is no improvement in the loss.

The training history can be seen in Figure 5.1.

### Results

Figure 5.2 shows the results achieved by CAE with a compression factor value of 4. The top row are the input patches belonging to the validation set. The bottom row are the outputs of the autoencoder.

The loss reached a plateau at $L = 0.0032$ which provide acceptable visual results. There is still some blurriness with respect to the original images. This could be improved in future versions of the network by tweaking the loss function, for example by adding a term which takes into account the differences between the gradients of the images.

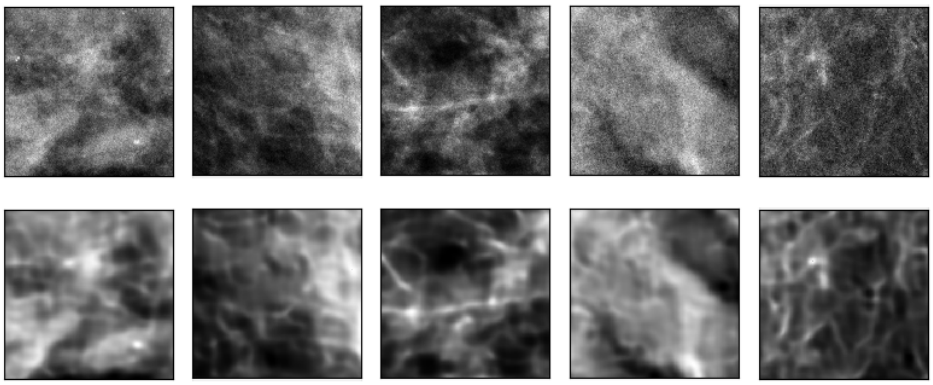

Figure 5.2: CAE results. Top row are the input images from the validation set. Bottom row are the reconstructed images generated by the autoencoder.

## 5.2 Convolutional Autoencoder + PCA

As described in Section 4.1.2, the purpose of adding a PCA proxy to the already trained CAE was to allow a higher degree of compression while maintaining the quality of the reconstructed image.

The layers of the PCA proxy were initialized based on the PCA projection of the data encoded using CAE. The size of the latent space corresponded to the number of principal components used for the projection. In particular we tried two different sizes, 100 and 1024, that respectively corresponded to an explained variance of 95% and 99%. We will refer to each network as CAE+PCA-1024 or CAE+PCA-100, the suffix indicating the size of the latent space.

The compression factor corresponding to these latent space sizes were $CF_{1024} = 64$ and $CF_{100} = 655$.

### Training

For training both CAE+PCA networks we first experimented with two strategies:

1. initializing the model with the CAE pretrained weights and the PCA parameters.

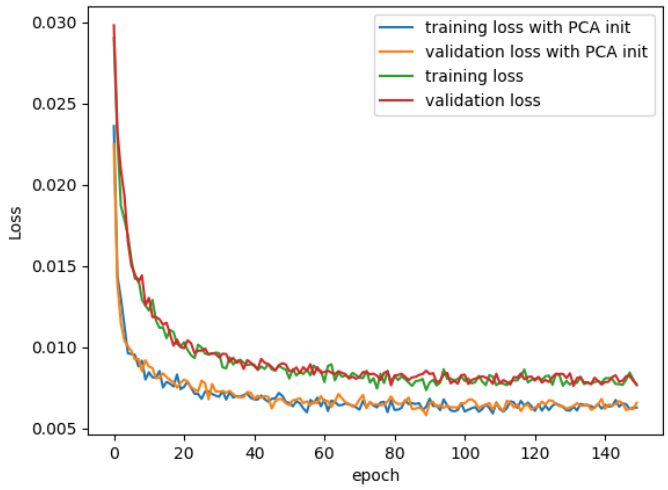


Figure 5.3: Experiments performed while training CAE+PCA-1024 to quantify the impact of using (bottom curves) or not using (top curves) as initial conditions for the network the PCA parameters.

2. initializing the model using only the CAE pretrained weights. The PCA proxy was initialized to random values.

The objective was to quantify the benefits of using the PCA parameters as initial conditions. The training configuration for both experiments was the same as the one used for optimizing CAE except for the number of epochs.

The training results for CAE+PCA-1024 are shown in Figure 5.3. Similar results were obtained for CAE+PCA-100. The difference between the two experiments proves that initializing the PCA proxy with the PCA parameters (bottom curve) not only speeds up the training but also improves the performance.

Based on this result, we trained CAE+PCA-100 and CAE+PCA-1024 using their respective PCA parameters as initial conditions. To improve performance even further we kept the PCA layers frozen during the first 50 epochs. The final loss achieved was $L_{100} \simeq 0.0065$ and $L_{1024} \simeq 0.0036$.

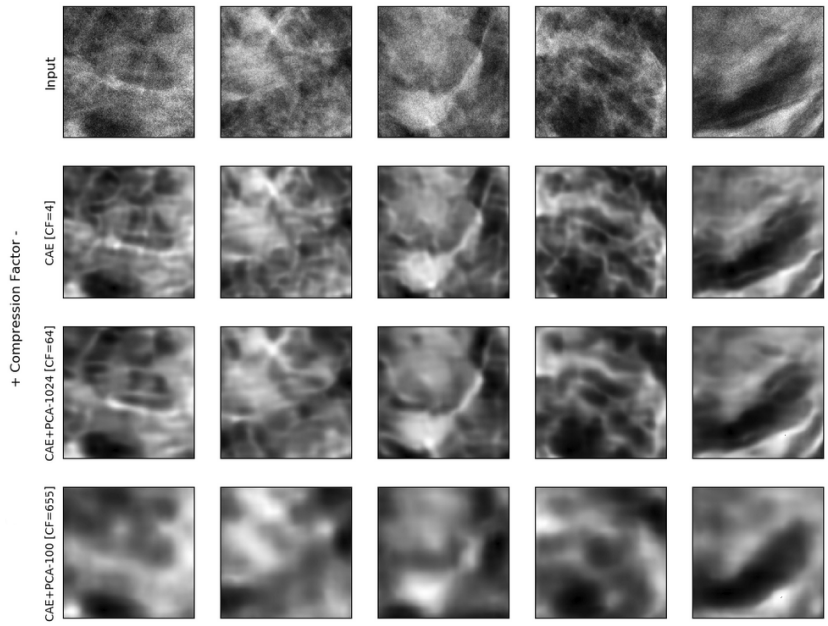


Figure 5.4: Reconstruction examples from the different networks we implemented: CAE, CAE+PCA-1024 and CAE+PCA-100. The first row contains the original images.

## Results

Figure 5.4 shows the results obtained with CAE+PCA-1024 and CAE+PCA-100. We also included the results from CAE to have a side by side comparison of all the networks implemented till now.

Overall, CAE+PCA-1024 was the best performing network considering the degree of compression, CF $= 64$, and the quality of the image, $L \simeq 0.0036$.

Furthermore, the quality downgrade between Input and CAE seems to be higher than between CAE and CAE+PCA-1024.

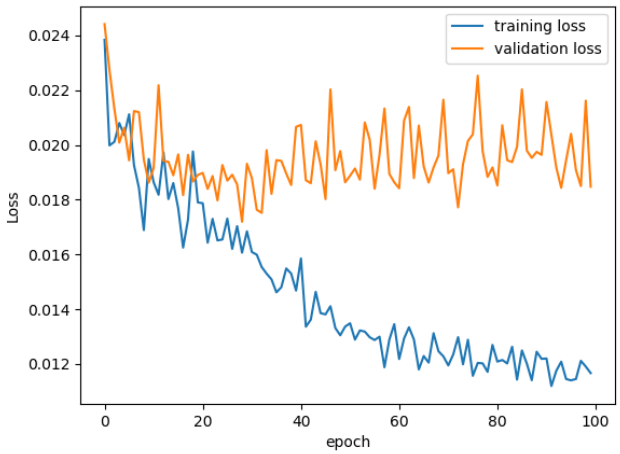


Figure 5.5: Asymmetric autoencoder training.

## 5.3 Asymmetric Autoencoder

As explained in Section 4.2.2, the Asymmetric Autoencoder uses as encoder the feature extraction layers from a DenseNet-121 trained on cancer recognition.

The idea was to test if the features learned for cancer classification were useful for image reconstruction. We also studied the general characteristics of the compressed representation.

The latent space was of size 1024, which means that $CF = 64$.

### Training

For training the Asymmetric Autoencoder, the first step was loading the Densenet-121 pretrained weights on to the network. Afterwards, like in most transfer learning applications, the transferred layers were kept frozen, meaning that their values were not updated during backpropagation. The idea was to keep the feature representations unchanged and verify if from them good image reconstruction was possible.

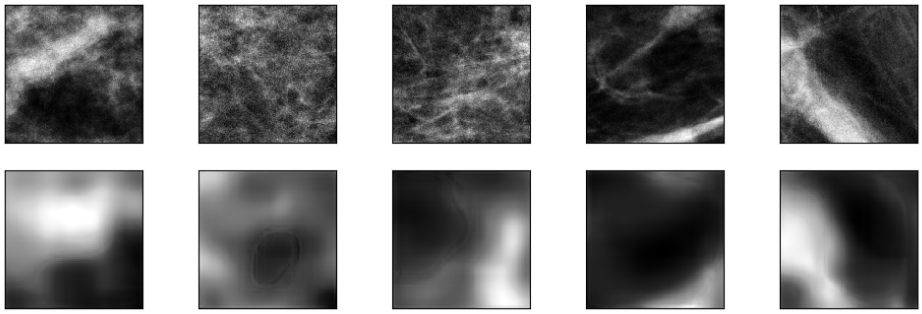

Figure 5.6: Asymmetric autoencoder results. Top row are the input images from the validation set. Bottom row are the output from the autoencoder.

The training protocol was the same as the previous experiments (see Section 5.1). Figure 5.5 shows the learning convergence. Note that the loss magnitude is an order of magnitude bigger than for the convolutional autoencoders.

### Results

The results achieved can be seen in Figure 5.6. If compared with CAE+PCA-1024, the results are clearly worst even though the compression factor is the same. This was not totally unexpected since the features extracted were not trained targeting an image reconstruction loss function. Results are in agreement with a study based on decoding neural network classifiers [22]. What is worth noticing is that the information containing the spatial location of objects within the image is still present, even if the output image is drastically blurred.

One way of improving the performance is to unfreeze some layers from the encoder, starting from the bottleneck. This will probably improve the reconstruction quality but will affect the nature of the features extracted.

On a last note, despite not containing all the information necessary for a faithful image reconstruction, the feature space generated by the DenseNet-121 is suitable for image comparison (see Section 4.2.1, particularly Figure 4.9). In the future, studying the way to merge the classifier's representation with the one obtained with CAE+PCA-1024 could result in an efficient and meaningful encoding.

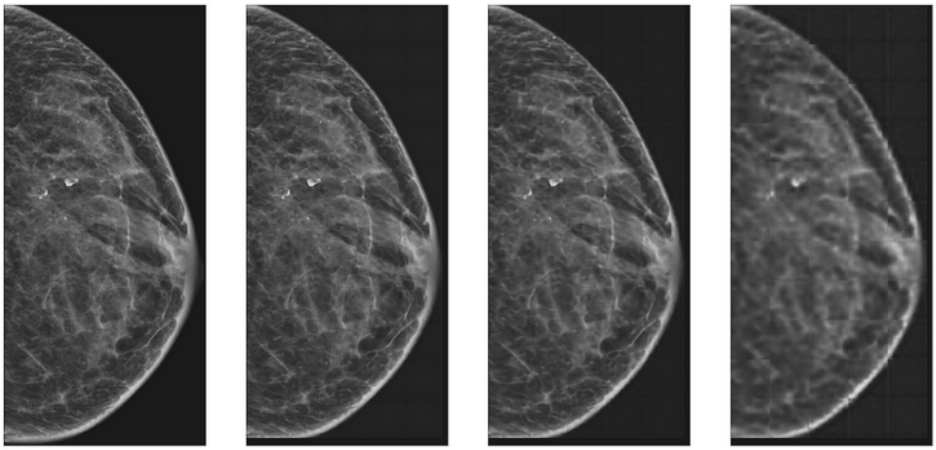


Figure 5.7: Reconstruction of a mammogram using the different versions of autoencoders we developed. The input image was an unseen image, namely we didn't use patches from this image to train or validate our models.

## 5.4 Full mammogram compression

As a proof of concept we ran a preliminary implementation of the full image compressor as described in Section 3.1, particularly Figure 5.7. We didn't include the version based on the Asymmetric Autoencoder because of its poor reconstruction performance.

As expected, the full image compressor with better performance is the one based on CAE+PCA-1024. To have an idea of orders of magnitude, the original image stored using the PNG format weights 10 MB. The compressed file weights 360 KB or 180 KB depending if using `float32` or `float16` to store the vectors representing each patch. Once again, this is a preliminary result but it serves as a reference for future work.

# 6 Deep Neural Network for breast-level cancer classification

One of the long term goals of the collaboration with CRO Aviano is to develop tools based on neural networks that assist clinicians with image interpretation, improving their diagnostic performance. This thesis represents a step forward in that direction. To reach this goal, it is important to be up to date with the latest developments in the field. The work described in this chapter was motivated exactly by this reason. In particular we used a state of the art deep convolutional neural network for breast cancer screening exam classification presented in March 2019 by Geras et al. [3]. The objective was to port their model to the platform of CRO Aviano as this provides a high-grade baseline for future work.

In the following, we give a general overview of the network's architecture and discuss the results obtained when tested using CRO Aviano's platform and dataset.

## Network overview

Figure 6.1 shows the architecture of the network from [3]. The model takes as input the four views of a mammogram exam, L-CC, R-CC, L-MLO and R-MLO, and outputs the probability of a benign or malignant finding for the left and right breast separately.

Although complex at first glance, the architecture has the two core modules every classification network has (Figure 4.7): a feature extraction module and a classification module. The feature extraction module is composed by four columns based on the ResNet architecture [23], a state of the art network used in computer vision problems. Each column takes as input one of the four views of a mammogram exam. In order to exploit the symmetry between the left and right breast, the columns applied to the L-CC/R-CC views share their weights.

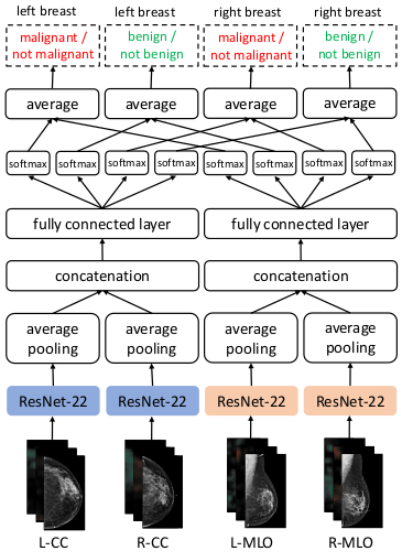


Figure 6.1: Schematic of the breast-level cancer classification network as presented by Geras et. al [3].

The same holds for the L-MLO/R-MLO views. The output of each column, after an average pooling operation, is a feature vector of size 256.

The classification module is composed by a CC and a MLO column, which respectively takes as input the concatenated L-CC/R-CC and L-MLO/R-MLO representations, each a vector of size 512. Both columns operate in the same way. The concatenated CC (MLO) representation is fed to two fully connected linear layers which generate a prediction for the four outputs. Both independent predictions are then averaged to obtain the final output.

In an alternative version of the network, malignant and benign heatmaps of each view are added as inputs. The heatmaps are generated by applying a patch-level cancer classifier, DenseNet-121, in a sliding window fashion over the full images. In this way, the authors claim to achieve the best performance, as they are exploiting the information provided by a fine grain model with the one provided by the breast-level model.

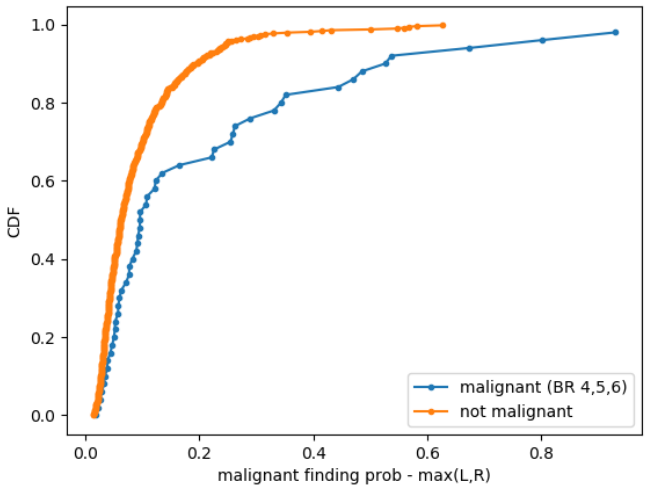


Figure 6.2: CDF calculated on the malignant prediction probability for both BI-RADS malignant cases (blue curve) and BI-RADS benign or healthy cases (orange curve). The malignant prediction was considered to be the maximum score obtained by either the left or right breast.

## Results

To port the deep neural network classifier to CRO Aviano's platform we created a small Python interface. The interface is based on a routine that allows the model implementation to accept mammograms stored in `DICOM` format instead of `PNG` format. It also generates an additional metadata file required to run the model. The file contains an entry per exam with the following tags stored as a dictionary:

```
{
'horizontal_flip': 'NO',
'L-CC': ['0_L_CC'],
'R-CC': ['0_R_CC'],
'L-MLO': ['0_L_MLO'],
'R-MLO': ['0_R_MLO'],
}
```

To evaluate the accuracy of the classifier on CRO Aviano's dataset we used the

BI-RADS labels as ground truth 2.1. In particular we will target the performance in predicting malignant findings (BI-RADS$\geq$ 4) as it is the category for which the model performs better.

Figure 6.2 shows the cumulative distribution function (CDF) calculated on the malignant prediction for both BI-RADS malignant cases (blue curve) and BI-RADS benign or healthy cases (orange curve). Since the BI-RADS labels referred to the whole exam, without left or right breast distinction, the malignant prediction was considered to be the maximum score obtained for either the left or right breast.

From the results it is clear that the model is capable to distinguish the two classes but only approximately. In fact the area under ROC curve (AUC) for these results was of 0.65; significantly better than random but still not very accurate, especially when compared with the AUC of 0.89 achieved by the authors on their dataset.

The main reason for these poor results is the nature of the BI-RADS labels; they correspond to a confidence level of the radiologist which later can be proven wrong or right. Furthermore radiologists uses different studies to define the label, not only mammographies. This means, for example, that their assessment may be based on a lesion seen on an ultrasound or MRI exam but not visible on mammography.

In the future, using pathology reports from biopsies to obtain the ground truth may enable obtaining more accurate results.

# 7 Conclusions

In this thesis we studied the use of neural networks for building a compression algorithm focused on mammogram images. The compressor had not only to be efficient but also to generate a reduced representation that captured image semantics, enabling smart image comparison.

The first step of the project was data preparation. CRO Aviano provided 1050 mammogram exams with their corresponding radiology reports which needed to be filtered and organised. Simple scripts were developed for this task, resulting in a curated dataset containing 662 exams, for a total of 2648 mammograms.

The amount of samples and the size of the images, $2560 \times 3328$, were not suitable for experimenting with neural network models, thus we created a new patch-level dataset based on the previous one. This second dataset consisted of approximately 70000 patches of size $256 \times 256$ which were randomly extracted from full size images. An ad hoc Python package was developed for this task, `mm_patch` [4], which includes simple data filtering options in addition to a flexible interface which allows adjusting different parameters of the extraction mechanism: size of the patches, maximum amount of patches extracted per image, overlapping versus non overlapping extraction, etc.

Once the data was ready, we moved on to study different strategies for building the compression algorithm. The general idea was to analyse the compression performance of different types of neural networks at a patch level, which can then be applied by blocks over the full image. In particular we analysed three different versions of neural network autoencoders, distinguished by their feature extraction mechanism and degree of compression achieved.

The first algorithm implemented was a 10 layered Convolutional Autoencoder (CAE) which achieved a small compression factor of 4 and produced high quality reconstructed images (Figure 5.2).

The second network was based on the first one. The idea was to increase the degree of compression by introducing two additional linear layers which acted as a Principal Component Analysis (PCA) proxy, thus its name: Convolutional Autoencoder plus PCA. The key of this implementation was to initialized those layers using the parameters of an actual PCA transformation performed on the image representations of the trained CAE. Using this technique we achieved a compression factor of 64 while maintaining the image reconstruction quality at similiar levels (Figure 5.4).

The third autoencoder we analysed, the Asymmetric Autoencoder, was intrinsically different from the previous ones. It used as an encoder the feature extracting layers of a DenseNet-121 classifier trained on breast cancer recognition. The image reconstruction results obtained with this architecture were poor (Figure 5.6), the problem being that the features learned for the classification were not sufficient to reconstruct the image. Although not useful as a compression mechanism, the analysis of the image representations generated by DenseNet-121 showed that the euclidean distance in the compressed space correlates with image similarities as human perception 4.9. This characteristic was not observed in the latent space generated by the others autoencoders.

This last result is really important for future developments; merging the Convolutional Autoencoder plus PCA representation with the DenseNet-121 representation has the potential of enabling a highly efficient mammogram compression algorithm with an encoded representation suitable for image queries at a visual semantic level. The techniques described could also be extended to different types of medical images.

# Bibliography


[1] T. Murdoch and A. Detsky. "The inevitable application of big data to health care." In: *JAMA* (2013), pp. 1351–52 (cit. on p. 1).

[2] A. Jemal et al. "Global Cancer Statistics 2018: GLOBOCAN Estimates of Incidence and Mortality Worldwide for 36 Cancers in 185 Countries." In: *CA: A Cancer Journal for Clinicians* (2018). DOI: 10.3322/caac.21492 (cit. on p. 2).

[3] Wu N, Phang J, Geras KJ. et al. "Deep Neural Networks Improve Radiologists' Performance in Breast Cancer Screening." In: *IEEE Transactions on Medical Imaging (T-MI)* (2019). DOI: 10.1109/TMI.2019.2945514 (cit. on pp. 3, 5, 25, 27, 41, 42).

[4] Federico Barone and Alessandro Laio. *Thesis repository*. 2019. URL: https://github.com/Fede112/patches (cit. on pp. 7, 32, 45).

[5] et al. Wu N. *The NYU breast cancer screening dataset v1.0*. 2019. URL: https://cs.nyu.edu/~kgeras/reports/datav1.0.pdf (cit. on p. 7).

[6] D. E. Rumelhart and J. L. McClelland. "Learning Internal Representations by Error Propagation." In: (1986). Ed. by MIT Press, pp. 318–362 (cit. on p. 14).

[7] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.A. Manzagol. "Stacked denoising autoencoders: Learninguseful representations in a deep network with a localdenoising criterion." In: *The Journal of Machine Learning Research* (2010) (cit. on p. 16).

[8] W. Wang, Y. Huang, Y. Wang, and L. Wang. "Generalized autoencoder: a neural network framework for dimensionality reduction." In: *Proceedings of the IEEE Conferenceon Computer Vision and Pattern Recognition Workshops* (2014), pp. 490–497 (cit. on p. 16).

[9] Q. Meng, D. Catchpoole, D. Skillicorn, and P. J. Kennedy. "Relational Autoencoder for Feature Extraction." In: *International Joint Conference on Neural Networks* (2017). DOI: 10.1109/IJCNN.2017.7965877 (cit. on p. 16).

[10] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng. "Measuring invariances in deep networks." In: *Advances in neural information processing systems* (2009), pp. 646–654 (cit. on p. 16).

[11] M. Mostajabi, M. Maire and G. Shakhnarovich. "Regularizing Deep Networks by Modeling and Predicting Label Structure." In: *Conference on Computer Vision and Pattern Recognition (CVF)* (2018) (cit. on p. 17).

[12] Lovedeep Gondara. "Medical image denoising using convolutional denoising autoencoders." In: *16th International Conference on Data Mining Workshops (ICDMW)* (2018). DOI: 10.1109/ICDMW.2016.0041 (cit. on p. 17).

[13] Cezanne Camacho. *Udacity Deep Neural Network Course*. 2018. URL: https://github.com/udacity/deep-learning-v2-pytorch/tree/master/autoencoder (cit. on p. 17).

[14] I.T. Jolliffe. *Principal ComponentAnalysis,Second Edition*. second edition. Springer, 2002 (cit. on p. 20).

[15] P. Baldi and K. Hornik. "Neural networks and principal component analysis: Learning from examples withoutlocal minima." In: *Neural networks* 2.1 (1989), pp. 646–654. DOI: https://doi.org/10.1016/0893-6080(89)90014-2 (cit. on p. 21).

[16] I. Sutskever A. Krizhevsky and E. Geoffrey Hinton. "Imagenet classification with deep convolutionalneural networks." In: *Advances in neural information processing systems* (2012), pp. 1097–1105 (cit. on p. 24).

[17] I. Laptev M. Oquab L. Bottou and J. Sivic. "Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks." In: *CVPR* (2014) (cit. on p. 24).

[18] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neuralnetworks?" In: *Advances in Neural Information Processing Systems* 27 (2014), pp. 3320–3328 (cit. on p. 24).

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cit. on p. 25).

[20] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. "Densely Connected Convolutional Networks." In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 4700–4708 (cit. on pp. 26, 27).

[21] Pablo Ruiz. *DenseNets guide*. 2018. URL: http://www.pabloruizruiz10.com/resources/CNNs/DenseNets.pdf (cit. on p. 27).

[22] A. Dosovitskiy and T. Brox. "Inverting Visual Representations with Convolutional Networks." In: *CVPR* (2015) (cit. on p. 39).

[23] S. Ren K. He X. Zhang. "Deep residual learning for image recognition." In: *CVPR* (2016) (cit. on p. 41).