SAPIENZA
UNIVERSITÀ DI ROMA

High-Resolution Image Segmentation and Classification in Breast Cancer Screening

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Magistrale in Artificial Intelligence and Robotics

Candidate

Federica Coppa
ID number 1749614

Thesis Advisors

Prof. Danilo Comminiello
Dr. Danilo Comminiello

Co-Advisor

Dr. Nome Cognome

Academic Year 2020/2021

Thesis defended on 16 April 2013
in front of a Board of Examiners composed by:

Prof. Nome Cognome (chairman)

Prof. Nome Cognome

Dr. Nome Cognome

High-Resolution Image Segmentation and Classification in Breast Cancer Screening

Master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: September 14, 2021

Author's email: coppa.1749614@studenti.uniroma1.it

Abstract

The aim of the project consists in presenting and analyzing the efficiency of an additional instrument in the fight against breast cancer: artificial intelligence and, more specifically, deep learning. Due to the fact that breast cancer is one of the major causes of death among women, many solutions to this problem have been proposed. However, artificial intelligence represents a foremost help for radiologists. In particular two approaches are analyzed: image classification and, in a second moment due to some problems related to image classification, pixel segmentation. This second method will give as a result fondamental information that, combined with the work of experts can improve the prevention of breast cancer.

Structure of the thesis

The following work is structure in five main points.
The first gives a complete panoramic about the breast cancer problematic and explain, in syntesis, the reasons why deep learning is so important.
A second section where CNN for image classification are presented with information about the architectures. In fact image classification is the first method
Then, the innovation approach is presented. It consist in the image segmentation of mammograms in order to highlight the mass related to the tumors, no matter if it is benignant or not. It can be an innovative important tool for riadiologists ′ from the moment that combining it with experts ′ diagnosis a more complete scenario is obtained.
The forth section is related to the analysis of the results obtained. Both for image classification and image segmentation.
Last, a short conclusion closes the project.

Contents

Chapter 1

Introduction

1.1 Breast cancer: problematics and tools

Breast cancer nowadays is one of the major causes of death among women. Every year, 12% of women are diagnosed with breast cancer (McGuire et al., 2015). In the US alone, 40,000 women die of breast cancer annually (Bray et al., 2018). Evidence shows that early detection of breast cancer can significantly increase the survival rate of women. In fact *Global Cancer Observatory Database (GLOBOCAN)* released in the year 2018, states that breast cancer is contributing to over 25.4% of all the new cases diagnosed (Gonella et al., 2020).
In order to detect and face this issue many tools and techniques are available. First of all screening programs are organized to aid in the early diagnosis of the disease. Mammograms are widely considered as the primary screening tool for breast cancer. Mammography has a central role in screening and diagnosis, since it allows early detection of the pathology and reduction of fatal cases. A mammogram is an X-ray image of the breast, which capture the changes in the breast tissue.
For this reason it is important to spend some words about breast tissue to understand its importance with respect breast cancer. Breast tissue (Boyd et al., 2010; Urbaniak et al., 2016) is composed of milk glands, milk ducts and supportive tissue (dense breast tissue), and fatty tissue (nondense breast tissue). When viewed on a mammogram, women with dense breasts have more dense tissue than fatty tissue. On a mammogram, nondense breast tissue appears dark and transparent. Dense breast tissue appears as a solid white area on a mammogram, which makes it difficult to see through. The radiologist who analyzes mammogram determines the ratio of nondense tissue to dense tissue and assigns a level of breast density. Breast density is a measure used to describe the proportion of fibroglandular tissue in a woman ′s breast depicted on a digital mammogram. Breast density can be measured *qualitatively or quantitatively.* Qualitative methods include the original Wolfe criteria, the Tabar classification, and the widely used Breast Imaging and Reporting Data System (BI-RADS) criteria. (Mohamed et al., 2018) The BI-RADS *Breast Imaging Reporting and Data System (BI-RADS)* mammographic breast density criteria include four qualitative categories illustrated in 1.2:

- A: Almost entirely fatty indicates that the breasts are almost entirely composed of fat. About 1 in 10 women has this result

- B: Scattered areas of fibroglandular density indicates there are some scattered areas of density, but the majority of the breast tissue is nondense. About 4 in 10 women have this result.
- C: Heterogeneously dense indicates that there are some areas of nondense tissue, but that the majority of the breast tissue is dense. About 4 in 10 women have this result.
- D: Extremely dense indicates that nearly all of the breast tissue is dense. About 1 in 10 women has this result.

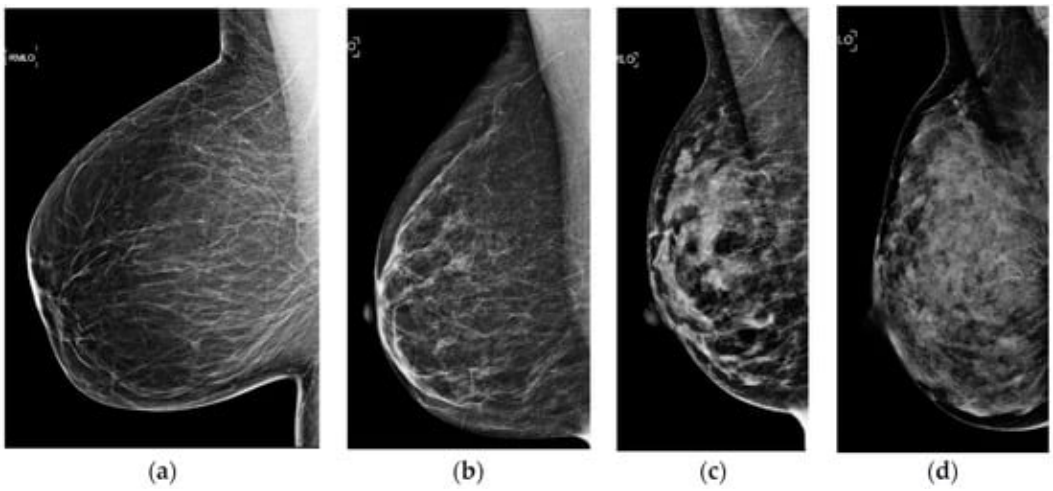

Figure 1.1. Breast density (a) almost entirely fatty, (b) scattered fibroglandular tissue, (c) heterogeneously dense, and (d) extremely dense, as determined by the BI-RADS Atlas.

In general, women with breasts that are classified as heterogeneously dense or extremely dense are considered to have dense breasts. About half of women undergoing mammograms have dense breasts. It's not clear why some women have a lot of dense breast tissue and others do not. However it is possible that having dense breasts is correlated to the factors explained below:

- *being young.* Breast tissue tends to become less dense as woman age, though some women may have dense breast tissue at any age.
- *lower body mass index.* Women with less body fat are more likely to have more dense breast tissue compared with women who are obese.
- *hormone therapy for menopause* Women who take combination hormone therapy to relieve signs and symptoms of menopause are more likely to have dense breasts.

All these description in details about breast tissue matters, since having dense breasts implies two main conseguences, it increases the chance that breast cancer may go undetected by a mammogram, as dense breast tissue can mask a potential cancer

and it increases risk of breast cancer.
Another factor to consider when detect breast cancer is that presence of mass and microcalcification in the mammograms characterize the disease. The detection of these regions is difficult since their pixel intensities often correlate with the normal tissue.
During a mammographic examination, the angles of image acquisition are 2 for each breast and they are shown and compared in 1.2

- bilateral craniocaudal (CC)
- mediolateral oblique (MLO)

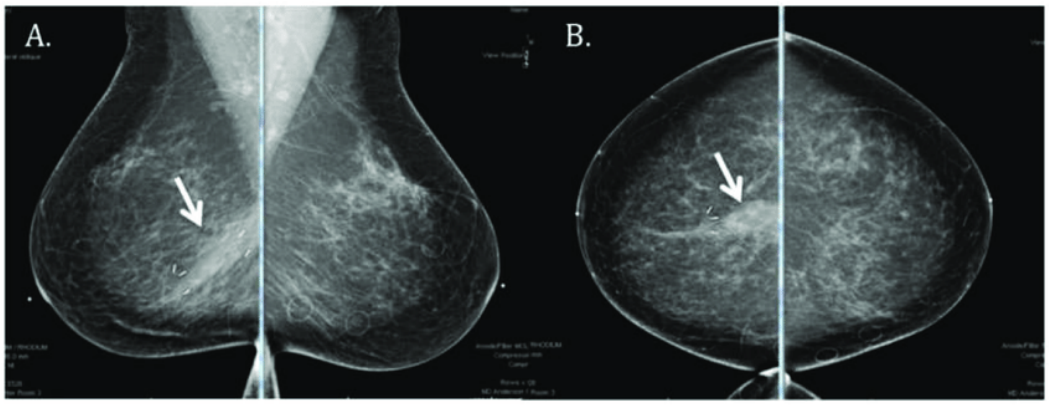

Figure 1.2. A) medio-lateral oblique and (B) cranio-caudal views

The visual inspection of these images should enable radiologists to examine and identify tissues carefully, looking for malignant lesions. Malignant lesions are mainly in the form of clusters of microcalcifications and masses. Unfortunately, the evaluation of mammograms is a complicated and time-consuming process that inevitably affected by high intra- and inter-observer variability.
Other tools are *Computer-aided detection* and *computer-aided diagnosis* that are known as CAD systems; they are used to lessen the burden of the radiologist in the process of diagnosis. They interpret digitally captured medical images and provide useful information about the suspicious regions. They employ artificial intelligence and image processing techniques for the task of detection and analysis. CAD systems are constantly evolving with the advent of new techniques in the domain to provide accurate results.

1.2 Deep learning and CNN

CNNs were used to face the problem of breast cancer in 1995 (Chan et al., 1995) and the use of Deep Learning for medical images is dated back to 1940s.
Deep neural networks usually have several hidden layers in between input layer and

output layer. These networks are used to retrieve features from images, whereas traditional ML algorithms use hand-engineering features for breast cancer detection. A newtype of deep learning (DL) is machine learning neural networks (ML-NNs), with ML-NN structures mostly requiring a training stage to find optimum weights. The most frequently used learning rule is the **backpropagation algorithm** in which weights are updated systematically for every pass depending upon the error rate obtained from the production layer with the gradient and chain rule. Deep Learning is a subset of machine learning that requires a huge number of labeled data to train the models. The term deep usually indicates the number of hidden layers in neural networks,for example ResNet has a depth of 152 layer which is 8× deeper than VGG-Net. Since 2012, CNNs have become more popular and have attracted more attention because of the increasing computing power, availability of lower cost hardware, open source algorithms, and the rise of big data.
Famous CNNs such as Alex-Net (Krizhevsky, 2014), ZF-Net(Shih et al., 2019), GoogLeNet (Al-Qizwini et al., 2017), VGG-Net (Sengupta et al., 2019) and ResNet (He et al., 2015) have brought about breakthroughs in processing images. Alex-Net architecture is extensively used in medical imaging for breast cancer detection. The structure of CNNs is very similar to that of ordinary neural networks.
CNNs are applied to explore patterns in an image (Alanazi et al., 2021). This is done by convoluting over an image and looking for patterns. These network can detect lines and corners in the few front layers of CNNs. neural net, however, we can then transfer these patterns down and begin to identify more complex characteristics as we get deeper. This property implies that CNNs are very effective at detecting objects in images.
The proposed system uses CNNs to detect breast cancer from breast tissue images. The basic CNN architecture illustrated in 1.3 is a stack of convolutional layer (*Conv*), nonlinear layer (for example *ReLU*), pooling layer (for example *Max-pooling*), and a loss funtion like *SVM or Softmax* on the last fully connected (*FC*). The output can be a single class, in the case of the problem of breast cancer maybe normal, benign, malignant or a probability of classes that best describes the image.

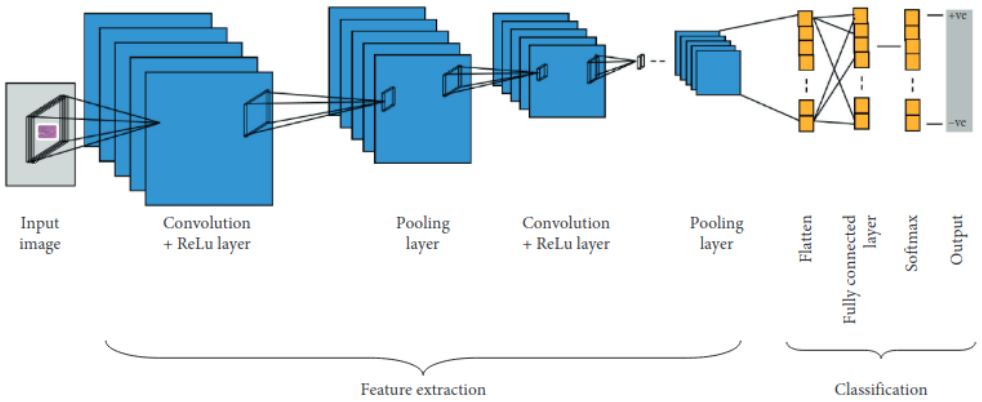

Figure 1.3. CNN architecture for automatic detection of breast cancer (link)

Now it is important to understand the importance of Deep Learning with respect breast cancer problematic. The use of breast density as a proxy for the detailed information that mammogram leads to, is limited because breast density assessment is a subjective assessment and varies widely across radiologists (Yala et al., 2019), and breast density summarizes the information contained in the digital images into a single value. Same age patients who are assigned the same density score can have drastically different mammography with vastly different outcomes. The problem is that there may are subtle but very informative cues on mammograms that may not be discernible by humans and deep learning can leverage these cues to yield improved risk models.

Chapter 2

Deep learning and architectures

Deep neural networks and deep learning are powerful and popular algorithms. A lot of their success lays in the careful design of the neural network architecture. In particular in this chapter six important architectures will be described, since they were chosen to performe the **image classification**, the first method presented in this project. Before doing that, it is important to give a general view of CNN and deep learning for image classification.
Image Classification is the process of extracting information from the images and labelling or categorizing the images. There are two types of classification:

- binary classification
- multi-class classification

Binary classification is a type of classification where the output is in binary value either 0 or 1, and as it will be explained it is the type of classification related to the first part of this project where benignant tumors are divided from the malignant ones.
multi-class classification is the type of classification where the output will be multi-class. In general, for image classification the architectures that are used mainly are LeNet-5, AlexNet, VGG, Inception and GoogLeNet, Residual Network or ResNet.

2.1 AlexNet

The first architecture is **AlexNet**.In 2012, Alex Krizhevsky released AlexNet (Krizhevsky, 2014) which was a deeper and much wider version of the LeNet and won by a large margin the difficult ImageNet competition.
AlexNet scaled the insights of LeNet, the very first convolutional neural networks, and what propelled the field of Deep Learning, into a much larger neural network that could be used to learn much more complex objects and object hierarchies. The contribution of this work are listed below:

- the use of rectified linear units **(ReLU)** as non-linearities;
- use of dropout technique to selectively ignore single neurons during training, a way to avoid overfitting of the model;

- overlapping max pooling, avoiding the averaging effects of average pooling;
- use of GPUs NVIDIA GTX 580 to reduce training time;

At the time GPU offered a much larger number of cores than CPUs, and allowed 10× faster training time, which in turn allowed to use larger datasets and also bigger images.
The success of AlexNet started a small revolution. Convolutional neural network were now the workhorse of Deep Learning, which became the new name for large neural networks that can now solve useful tasks. As explained in (Krizhevsky, 2014) AlexNet consists of eight layers: five convolutional layers and three fully-connected layers. The architecture is represented in 2.1:

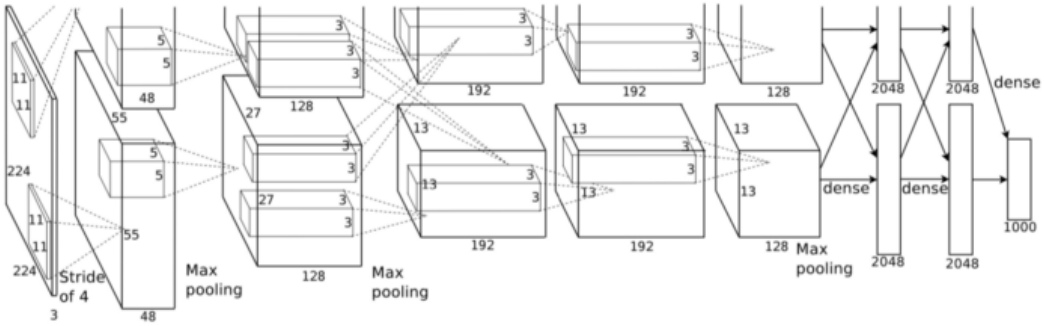

Figure 2.1. architecture of *AlexNet* (link)

However the most important peculiarities of AlexNet are three and they are illustrated in 2.2:

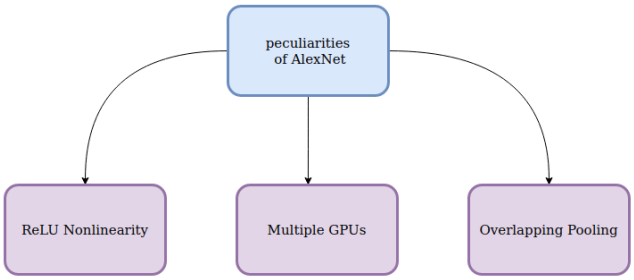

Figure 2.2. peculiarities of *AlexNet*

About ReLU Nonlinearity, AlexNet uses **Rectified Linear Units (ReLU)** instead of the *tanh* function, which was standard at the time. ReLU′s advantage is in training time.

About multiple GPUs, GPUs at first were still rolling around with 3 gigabytes of memory. This was especially bad because the training set had 1.2 million images. AlexNet allows for multi-GPU training by putting half of the model′s neurons on one GPU and the other half on another GPU. It implies two main consequences that are schematized in 2.3 :

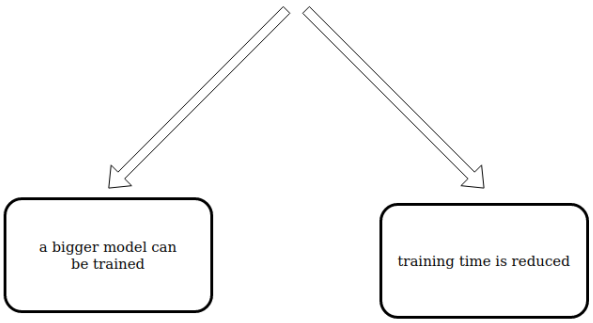

Figure 2.3. consequences of using multiple GPUs in *AlexNet*

The advantages of using a multiple GPUs in *AlexNet* are important and they are clarified in 2.4 as follows:

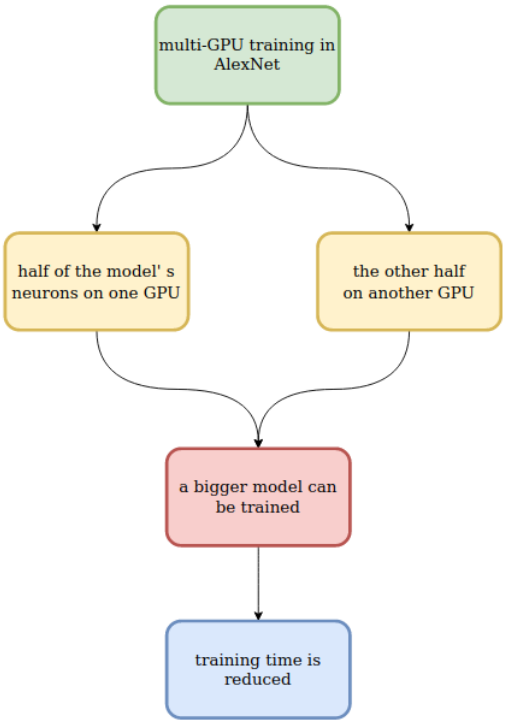

Figure 2.4. Advanatges of using multiple GPUs in *AlexNet*

About Overlapping Pooling, CNNs traditionally pool outputs of neighboring groups of neurons with no overlapping. However, when overlap is introduced, a reduction in error by about 0.5% is noticed and found that models with overlapping pooling generally find it harder to overfit.
In addition AlexNet had 60 million parameters, a major issue in terms of overfitting. Two methods were employed to reduce overfitting:

- Data Augmentation;
- Dropout;

Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularizer and helps reduce overfitting when training a machine learning model.
In (Srivastava et al., 2014) is explained what **dropout** is and why it is so important. In few lines it is possible to highlight its most important peculiarities and what it means. **Dropout** consists in *turning off* neurons in a random way. They are temporarily removed from the network and the remaining neurons are forced to work also for the others. This process is repeated randomly and it can be visualized in 2.5 as follows:

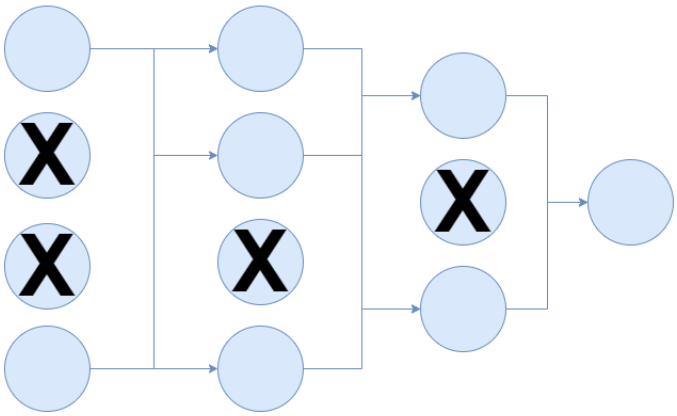

Figure 2.5. Representation of dropout

In figure 2.5 it is possible to see that every iteration uses a different sample of the model′s parameters, which forces each neuron to have more robust features that can be used with other random neurons. However, a disadvantage aspect of dropout is that it also increases the training time needed for the model′s convergence.

2.2 ResNet

The second architecture analized is **ResNet-18**. As explained in He et al. (2015), when the network depth is increased the accuracy gets saturated and then degrades rapidly. By observing the training errors of various network depths, it is evident that the degradation is not caused by overfitting. The **training accuracy degradation** indicates that the deeper network architectures are harder to optimize due to **vanishing/exploding gradients**. While training, the vanishing gradient effect on network output with regard to parameters in the initial layer becomes extremely small. The gradient becomes further smaller as it reaches the minima. As a result, weights in initial layers update very slowly or remain unchanged, resulting in an increase in error. **Residual networks** addressed the degradation problem by implementing skip con nections. It has been hypothesized that it is better to optimize the network with **skip connections** than to optimize the original. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers.
A residual network, ResNet for short, is an artificial neural network that helps to build deeper neural network by utilizing *skip connections* or *shortcuts* to jump over some layers. There are different versions of ResNet: ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152, ResNext-50_32x4d, ResNext-101_32x8d, wide_ResNet-50_2, wide_ResNet-101_2.

The numbers denote layers, although the architecture is the same. There are two main types of blocks used in ResNet, depending mainly on whether the input and output dimensions are the same or different.

- Identity Block;

- Convolution Block;

Identity Block is when the input and output activation dimensions are the same. **Convolution Block** is when the input and output activation dimensions are different from each other.

The convolutional layers mostly have 3×3 filters and follow two simple design rules as explained in 2.6

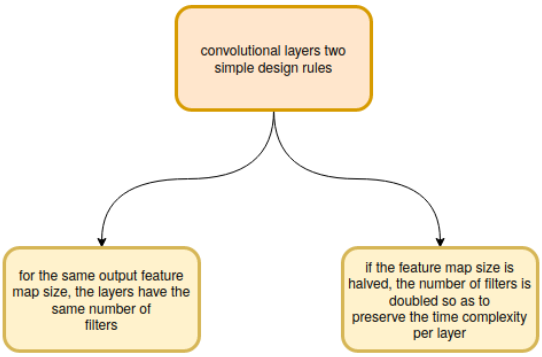

Figure 2.6. two simple design rules for convolutional layers

Downsampling is performed directly by convolutional layers that have a stride equal to 2. The network ends with a global average pooling layer and a 1000-way fully-connected layer with softmax. The total number of weighted layers is 18 for ResNet-18. It is worth noticing that the model has fewer filters and lower complexity than VGG nets. In 2.7 it is possible to see the architecture of **ResNet-18**.

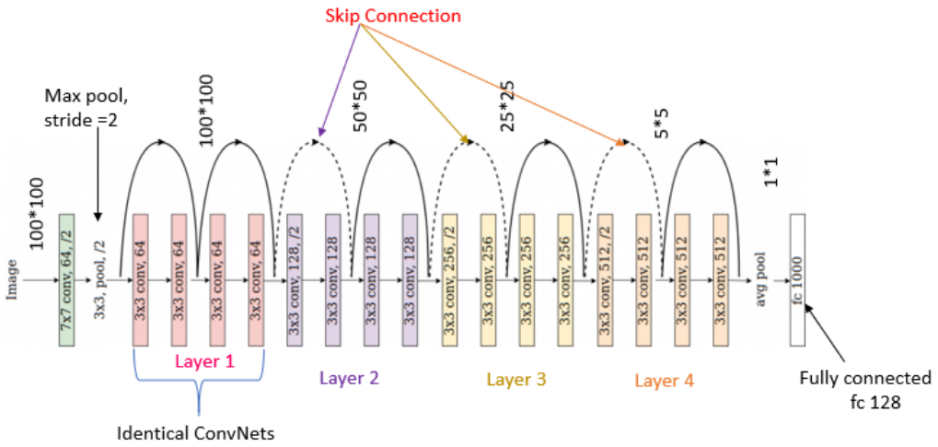

Figure 2.7. architecture of *ResNet-18* (link)

2.3 Densenet

Refering to (Huang et al., 2018) **Dense Convolutional Network, DenseNet**, which connects each layer to every other layer in a *feed-forward* fashion, is a type of convolutional neural network that uses dense connections between layers, through **Dense Blocks**, where we connect all layers directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Whereas traditional convolutional networks with L layers have L connections, one between each layer and its subsequent layer, **DenseNet** has $L(L+1)/2$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. **DenseNets** have several compelling advantages that are represented in the scheme in 2.8 :

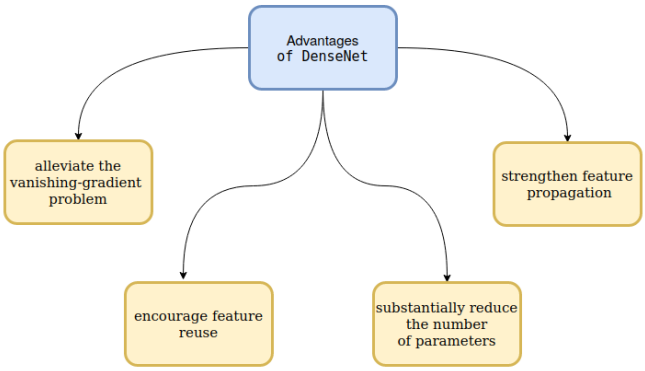

Figure 2.8. Advantages of DenseNet

DenseNet is composed of *Dense blocks*. In those blocks, the layers are densely connected together. Each layer receives in input all previous layers output feature maps. The architecture is reported in 2.9 as follows:

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 2.9. Structure of Densenet (link)

Since each layer receives more supervision from the loss function thanks to the shorter connections, **deep supervision** is created.
When talking about **DenseNet** it is necessary to introduce the concept of **dense block**. A **dense block** is a group of layers connected to all their previous layers. A single layer looks like this:

- Batch Normalization;
- ReLU activation;
- 3×3 Convolution;

An example of 5-layers dense block with growing rate of 4 is reported in 2.10 below:

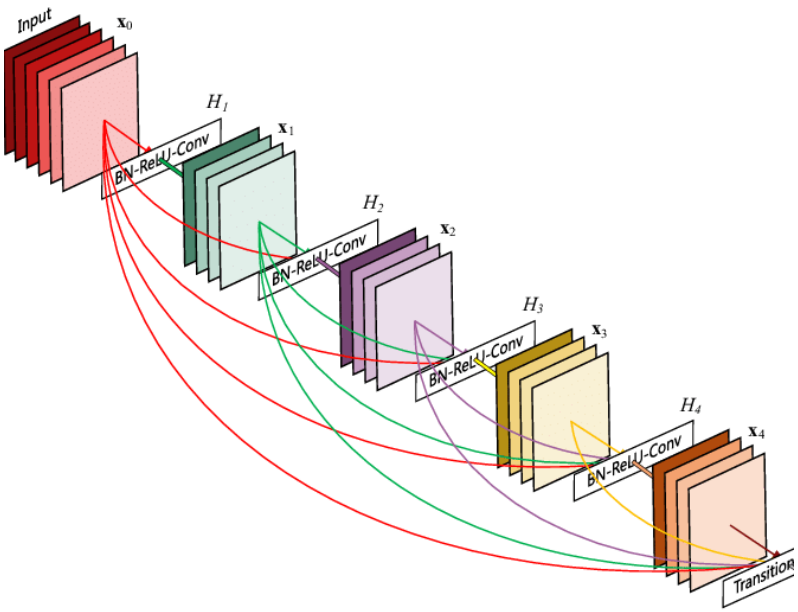

Figure 2.10. 5-layers dense block with growing rate of 4 (link)

Another important aspect is **transition layer**. Since DenseNet concatenates all the feature maps, it would be impracticable to concatenate feature maps of different sizes. Thus in each dense block, the feature maps of each layer has the same size. However down-sampling is essential to CNN. Transition layers between two dense blocks assure this role.

A transition layer is made of:

- Batch Normalization;
- 1×1 Convolution;
- Average pooling;

Another peculiarity of **DenseNet** is that it has less parameters than an usual SotA networks, for two main reasons:

- growth rate;
- bottleneck;

A DenseNetl′s convolution generates a low number of feature maps. The number of output feature maps of a layer is defined as the growth rate. DenseNet has lower need of wide layers because as layers are densely connected there is little redundancy in the learned features. All layers of a same dense block share a collective knowledge. In few words, The growth rate regulates how much new information each layer contributes to the global state.
each 3×3 convolution can be upgraded with a bottleneck. A layer of a dense block with a bottleneck will be constituited by:

- Batch Normalization;
- ReLU activation;
- 1×1 Convolution bottleneck producing grow rate multiplied 4 feature maps;
- Batch Normalization;
- ReLU activation;
- 3×3 Convolution;

Different type of **DenseNet** exists and they are represented in the scheme in 2.11

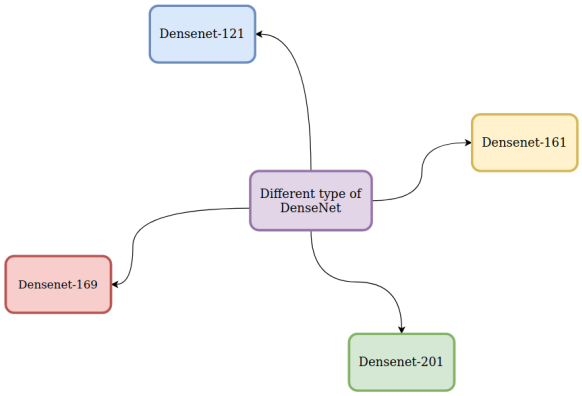

Figure 2.11. Different type of *DenseNet*

2.4 GoogLeNet and Inception

Christian Szegedy from Google started a search with the aim of reducing the computational burden of deep neural networks, and devised the GoogLeNet the first Inception architecture. By now, deep learning models were becoming extermely useful in categorizing the content of images and video frames. Most skeptics had given in that Deep Learning and neural networks came back to stay this time. Given the usefulness of these techniques, the internet giants like Google were very interested in efficient and large deployments of architectures on their server farms.

Christian thought a lot about ways to reduce the computational burden of deep neural nets while obtaining state-of-art performance (on ImageNet, for example). Or be able to keep the computational cost the same, while offering improved performance. He and his team came up with the Inception module 2.12:

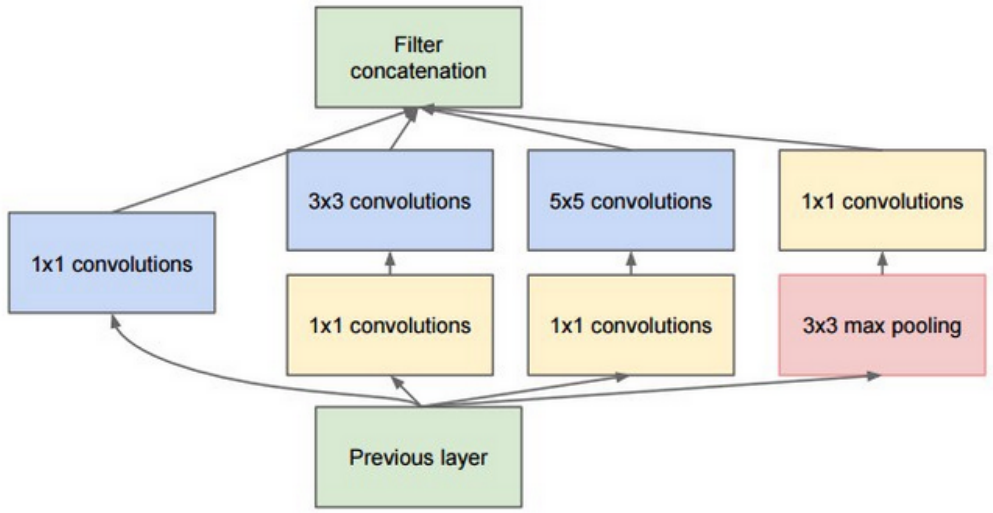

Figure 2.12. Inception module (link)

Inception module at a first glance is basically the parallel combination of 1×1, 3×3, and 5×5 convolutional filters. But the great insight of the inception module was the use of 1×1 convolutional blocks (NiN) to reduce the number of features before the expensive parallel blocks. This is commonly referred as bottleneck. GoogLeNet used a stem without inception modules as initial layers, and an average pooling plus softmax classifier similar to NiN. This classifier is also extremely low number of operations, compared to the ones of AlexNet and VGG. This also contributed to a very efficient network design. Specifically **GoogLeNet** is a deep convolutional neural network that is a variant of the **Inception Network**. GoogleNet is known also as **Inception V1** and was proposed by research at Google in 2014 in the research paper titled *Going Deeper with Convolutions*. It has provided a significant decrease in error rate as compared to previous architecture *AlexNet* and significantly less error rate than *VGG*.
The GoogLeNet architecture consists of 22 layers,27 layers if pooling layers are included and part of these layers are a total of 9 inception modules. The architecture is represented in 2.13 :

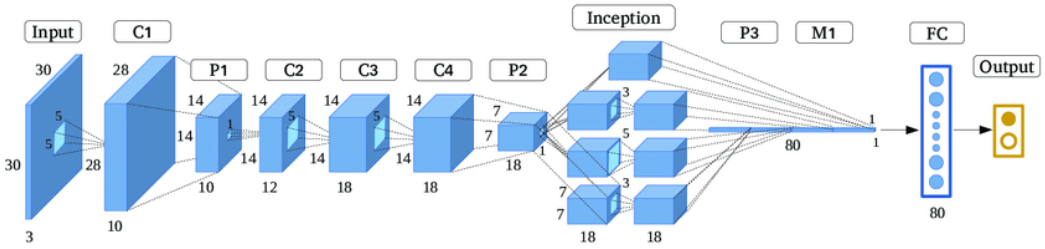

Figure 2.13. architecture of GoogleNet (link)

This architecture uses two important techniques as it is possible to see in 2.14 :

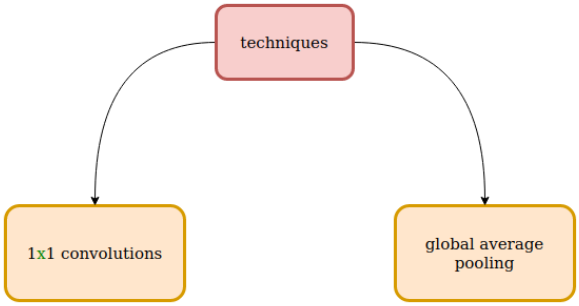

Figure 2.14. Characteristics of GoogleNet

Today GoogLeNet is very popular for some computer vision tasks as the scheme 2.15 shows:

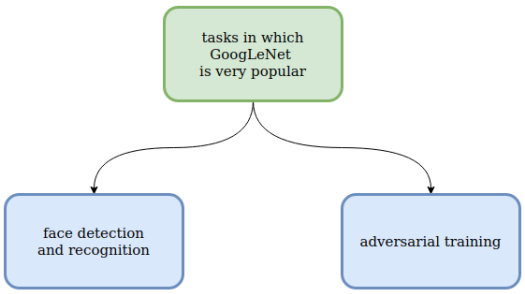

Figure 2.15. GoogLeNet and computer vision tasks

The GoogLeNet architecture is very different from previous architectures such as *AlexNet* and *ZF-Net*. Some comparisons with AlexNet and ZF-Net will be necessary in order to understand better GoogLeNet and its innovativity.
GoogLeNet uses many different kinds of methods such as 1×1 convolution and global average pooling that enables it to create deeper architecture. Now few words have to be spent for each of these characteristics:
1×1 convolution: the inception architecture uses 1×1 convolution. These convolutions are used to decrease the number of parameters (weights and biases) of the architecture. By reducing the parameters we also increase the depth of the

architecture.
Global Average Pooling: In other architecture for example AlexNet, the fully connected layers are used at the end of the network. These fully connected layers contain the majority of parameters of many architectures that causes an increase in computation cost. On the other hand in GoogLeNet architecture, there is a method called **global average pooling** used at the end of the network. This layer takes a feature map of 7×7 and averages it to 1×1. This also decreases the number of trainable parameters to 0 and improves the accuracy by 0.6%.

2.5 Inception

Another model that was tought to be used is **inception_v3**.From the paper (Szegedy et al., 2015), Inception_v3 is almost similar to inception_v2 except for some updates that will be discussed later. First some words about Inception_v2 and its properties and characteristics are needed in order to understand deeply inception_v3. Inception v2, 3×3 convolutions is used in the space of 5*times*5 and boosts the performance. This also decreases computational time and thus increases computational speed because a 5*times*5 convolution is 2.78 times more expensive than 3×3 convolution. For this reason, the use two 3×3 layers instead of 5×5 increases the performance of architecture.
The main differences between inception_v2 and inception_v3 are the use of RMSprop optimizer, batch normalization in the fully connected layer of the auxiliary classifier, also the use of 7×7 factorized convolution and *label smoothing regularization*. It is a method to regularize the classifier by estimating the effect of label-dropout during training. It prevents the classifier to predict too confidently a class. The main advantage of using label smoothing gives 0.2% improvement from the error rate.
The main characteristics of inception_v3 are illustrated in 2.16

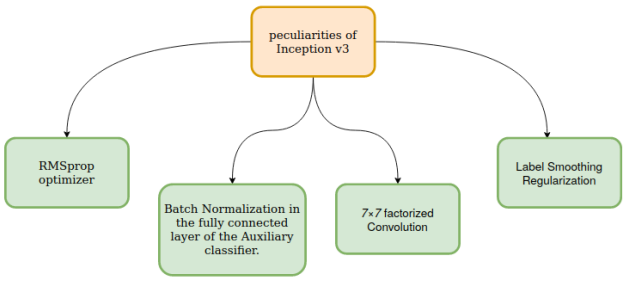

Figure 2.16. Peculiarities of inception_v3

The architecture is represented in the image 2.17:

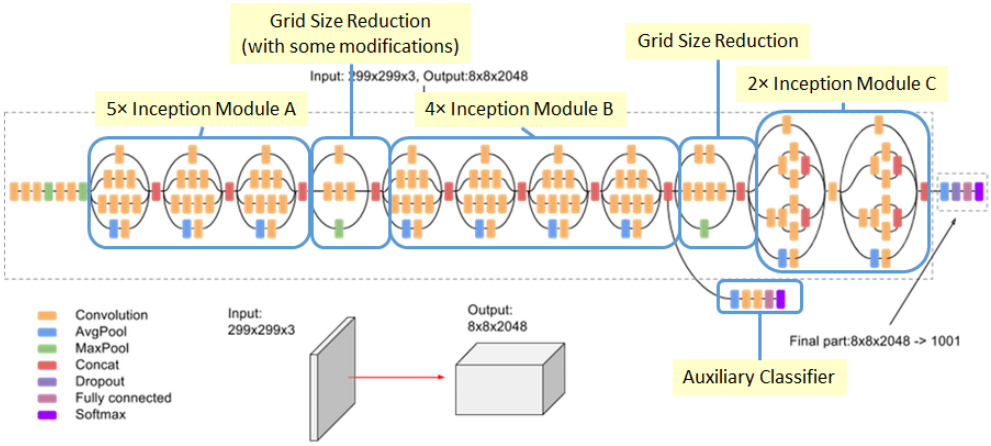

Figure 2.17. Architecture of inception_v3 (link)

2.6 ShuffleNet

As explained in (Ma et al., 2018), ShuffleNet architecture is composed of a stack of **ShuffleNet units** grouped into three stages. The ShuffleNet unit is presented in the image 2.18 and it is specially designed for small networks.

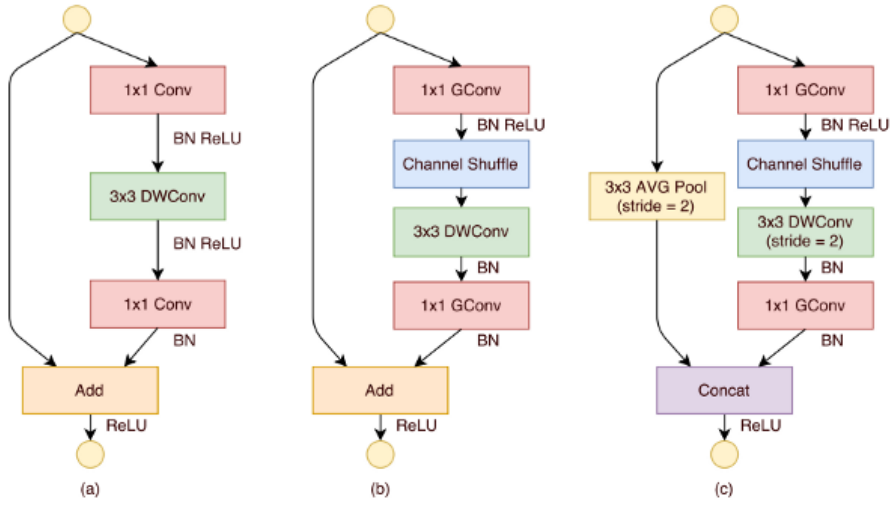

Figure 2.18. ShuffleNet unit (link)

In 2.18 it is possible to observe three different scenarios:

- a) a bottleneck unit with depthwise convolution;
- b) is a ShuffleNet unit with pointwise group convolution and channel shuffle;

- c) is a ShuffleNet unit with stride of 2;

Next, in order to understand better the architecture a table 2.19 is reported:

Table 1: ShuffleNet architecture

Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2[1]	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity[2]					143M	140M	137M	133M	137M

Figure 2.19. ShuffleNet architecture (link)

It is composed of a stack of ShuffleNet units grouped into three stages. In 2.19, the group number g controls the connection sparsity of pointwise convolutions. At the same time, g is assigned to be different numbers, so the output channels can be computed and evaluated to ensure the total computational costs are approximately the same.
ShuffleNet is widely adopted in low end devices such as mobiles and works well training on ImageNet. This influences the fact that the model does not performe well in the aim of this project, as it is possible to see later, analizing the plots.

Chapter 3

Innovative approach: pixel classification

In this thesis two tasks are considered on the dataset:

- image classification between benignant and malignant tumors;
- pixel classification to provide ROIs for a given radiology;

The **pixel classification** represents the innovative approach, which as it will be explained later, provides promising results. Before analyzing the results obtained, it is important to understand the basis and how pixel classification works and why this method appears so appropriate for the problem at hand.
According to what has been stated in the previous sections, an image classification problem did not give any good results on ROIs from CBIS-DDSM. After analized five responses from five different experts that studied and tried to stated a classification of 20 images from CBIS-DDSM ROIs, it was quite clear that this specific dataset was not appropriate for the problem. For this reason a solution to the problem was searched and it was decided to procede with a new type of problem, a *pixel classification* problem.
Before explaining the code used to this new problem and the results obtained it is important to spend some words about pixel classification, in order to understand why it is appropriate for the problem at hand. Image segmentation (Liu et al., 2021), or pixel classification, is a classic problem in computer vision research and has become a hotspot in the field of image understanding. The so-called image segmentation refers to the division of an image into several disjointed areas according to features such as grayscale, color, spatial texture, and geometric shapes. So that these features show consistency or similarity in the same area, but between different areas shows a clear difference. Image segmentation is divided into semantic segmentation, instance segmentation and panoramic segmentation according to the different coarse and fine granularity of segmentation. The field of interest right now is semantic segmentation. In fact segmentation of medical images is regarded as a semantic segmentation task. At present, there are more and more research branches of image segmentation, such as satellite image segmentation, medical image segmentation, autonomous driving. With the large increase in the proposed network structure, the image segmentation method is improved step by step to obtain more and more accurate

segmentation results. However, for different segmentation examples, there is no universal segmentation algorithm that is suitable for all images. Traditional image segmentation methods can no longer be compared with the segmentation methods based on deep learning in effect, but the ideas are still worth learning. Like the proposed threshold-based segmentation method, region based image segmentation method, and edge detection-based segmentation method. These methods use the knowledge of digital image processing and mathematics to segment the image. The calculation is simple and the segmentation speed is fast, but the accuracy of the segmentation cannot be guaranteed in terms of details. At present, methods based on deep learning have made remarkable achievements in the field of image segmentation. Their segmentation accuracy has surpassed traditional segmentation methods. The fully convolutional network was the first to successfully use deep learning for image semantic segmentation. This was the pioneering work of using convolutional neural networks for image segmentation.
Pixel classification approach (Hwang and Liu, 2015) also known as image segmentation, is characterized by two main disadvantages, in general terms. The first is the *high computational cost* that is an obvious consequence of the classification of each pixel in the input image. Second is the training of pixel classification algorithms can be problematic in the frontier of regions where near equivalent inputs to the classifier can correspond to different classes. The use of pixel classification is suitable when handling irregularly-shaped zones and when the feature extraction and classifier execution are not too much demanding from the computational point of view. It means this method is very appropriate for the problem of breast cancer, in particular when considering ROIs, since they are very irregular shape regions. In this technique a digital image is broken down into various subgroups called **image segments** which helps in reducing the complexity of the image to make further processing or analysis of the image simpler (Haralick and Shapiro, 1985; Pal and Pal, 1993). Segmentation in easy words is assigning labels to pixels. All picture elements or pixels belonging to the same category have a common label assigned to them. This task is a part of the concept of scene understanding or better explaining the global context of an image (Taghanaki et al., 2020). In the medical image analysis domain, image segmentation can be used for image-guided interventions, radiotherapy, or improved radiological diagnostics. This is another reason why the choice fell on this method in order to procede with this project.
Now it is possible to analize the impact of image segmentation applied on medical imaging. Image segmentation for medical images is based on the use of computer image processing technology to analyze and process 2D or 3D images to achieve what is schemed in 7.16:

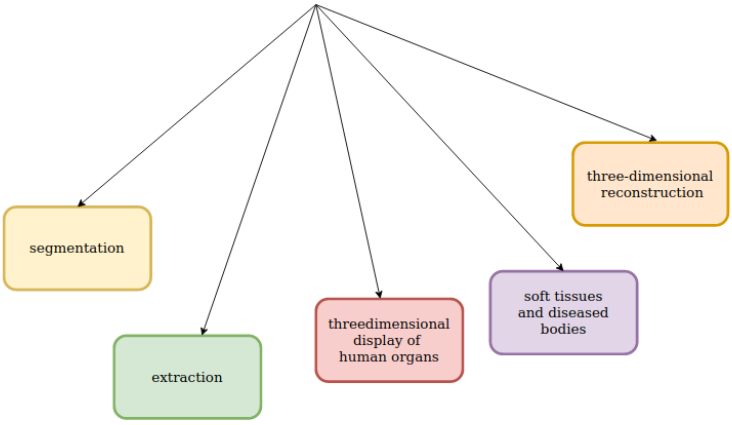

Figure 3.1. Aims of image segmentation for medical images

segmentation, extraction, three-dimensional reconstruction and threedimensional display of human organs, soft tissues and diseased bodies. It divides the image into several regions based on the similarity or difference between regions. Doctors can perform qualitative or even quantitative analysis of lesions and other regions of interest through this method, thereby greatly improving the accuracy and reliability of medical diagnosis. Currently, the main variety, tissues and organs of the image cells are used as object. Generally, medical image segmentation can be described by a set theory model: given a medical image I and a set of similarity constraints C_i (i = 1, 2, . . .), the segmentation of I is to obtain a division of it, namely:

$$\bigcup_{x=1}^{N} R_x = \mathrm{I}, \quad R_x \cap R_y = \varnothing, \quad \forall x \neq y,\ x, y \in [1, N]$$

Figure 3.2

where R_x satisfies both sets of all pixels in communication similarity constraint C_i (i = 1,2, . . .),the image areas. The same is true for R_y . x, y are used to distinguish the different regions. N is a positive integer not less than 2, indicating the number of regions after division. The process of medical image segmentation can be divided into the following stages:

- Obtain medical imaging data set, generally including training set, validation set, and test set. When using machine learning for image processing, the data set is often divided into three parts. Among them, the training set is used to train the network model, the verification set is used to adjust the hyperparameters of the model, and the test set is used to verify the final effect of the model.

- Preprocess and expand the image, generally including standardization of input image, perform random rotation and random scaling on the input image to

increase the size of the data set.

- Use appropriate medical image segmentation method to segment the medical image, and output the segmented images.
- Estimation performance evaluation. In order to verify the effectiveness of medical image segmentation, effective performance indicators need to be set to be verified. This is an integral part of the process.

Chapter 4

Data preprocessing

4.1 Choice of the dataset

The aim of this project is to analize and develop some techniques according to what has been explained above, in the hope to provide more solutions and tools to face breast cancer. Of course the process that lead to some results was not linear and easy. The first decision to make was choosing a dataset that seemed to be appropriate for this task.
In a first moment the datasets that have been take in consideration were the ones illustrated in 4.1 CBIS-DDSM, MIAS, INbreast, Breast Cancer Histopathological Database (BreakHis) and CSAW-S.

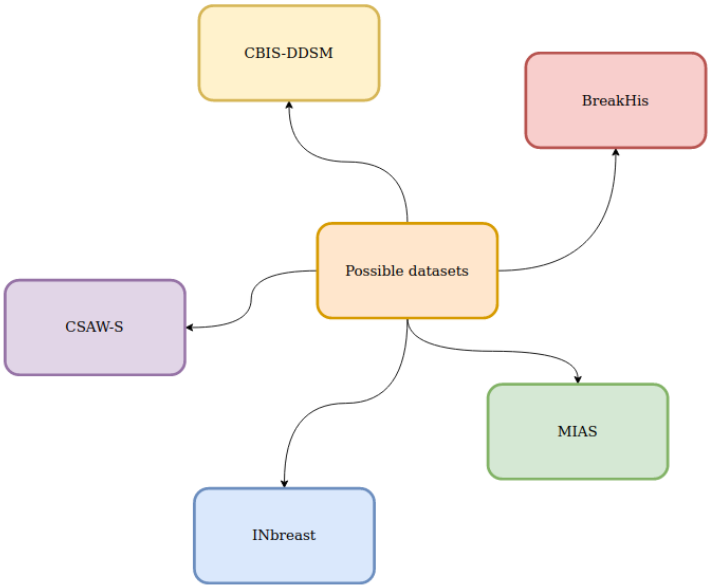

Figure 4.1. Five different medical datasets considered at the beginning of the project

Although not all of them are good for the aim of this project characteristics and properties of each one will be accurately described in order to have a clear idea of what it is required for this purpose.

4.1.1 CBIS-DDSM

The **CBIS-DDSM** (Rubin, 2018) **(Curated Breast Imaging Subset of DDSM)** dataset is an upgraded version of DDSM (Digital Database for Screening Mammography), which contains 10,239 processed mammography. The **CBIS-DDSM** is a significant public digitised screen-film mammography (SFM) database, including training and test cases of calcifications and masses. As explained in the introduction, detecting the presence of mass and microcalcification in the mammograms is important since they indicate the presence of a malformations. In fact CBIS-DDSM is characterized by only tumors images as they are divided in mass and calcifications. This is one of the main differences between CBIS-DDSM and DDSM. As it is explained at the end of this section DDSM contains also cases of non tumors mammography. In 4.2, it is possible to see the partition of **CBIS-DDSM**:

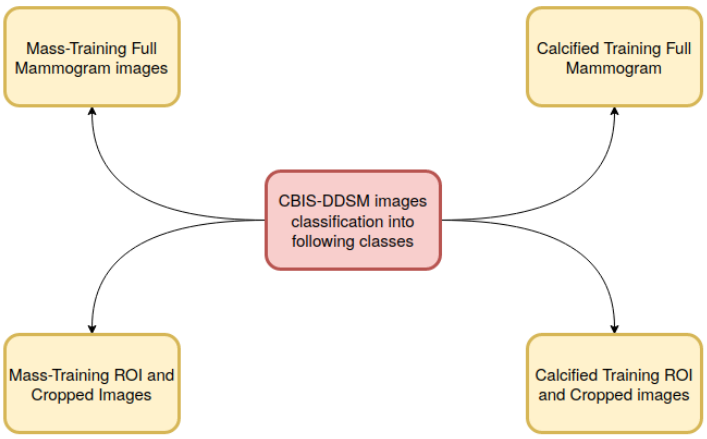

Figure 4.2. CBIS-DDSM images division into four classes

Along with the images there are two csv files, for Mass and Calcified training and test images.
Moreover, it includes both bilateral craniocaudal (CC) and mediolateral oblique (MLO) mammograms views, with an image size found equal to 3118x5001 for several image in DICOM format. Each case was reviewed by a mammogram, also performing an update of the ROI segmentations. **CBIS-DDSM** is a subset of **DDSM**. DDSM is a public mammogram database that is organized into cases. A case is a collection of mammograms corresponding to one patient taken from different views and angles. Each case belongs to one of these three pathologies:

- benign;

- malignant;
- normal;

It contains the patient′s information and a description of the mammogram, including the breast density rate (ACR), precise location, and the type of the abnormality. In addition, the benign and malignant cases are assoÂŋciated with their corresponding ground truth annotations.
The analysis of **DDSM** revealed some limitations, such as non-standard format and imprecise segmentation of the tumour area. To overcome these limitations, CBIS (Curated Breast Imaging Subset of DDSM) was used. In DDSM images were in *LJPEG* format and have been converted from **Lossless Joint Photographic Experts Group** (*LJPEG*) to **Digital Imaging and Communications in Medicine** *(DICOM)* format. During the conversion process, the images were decompressed, pixel values re-mapped to 16-bit grayscale.

4.1.2 MIAS

The **Mammography Image Analysis Society (MIAS)**, which is an organization of UK research groups interested in the understanding of mammograms, has produced a digital mammography databaseThe mini-MIAS database of mammograms (Debelee et al., 2019) The X-ray films in the database have been carefully selected from the United Kingdom National Breast Screening Programme and digitized with a Joyce-Lobel scanning microdensitometer, a device linear in the optical density range 0-3.2 and representing each pixel with an 8-bit word. The database contains left and right breast images for 161 patients, and is available on a DAT-DDS tape. Its quantity consists of 322 images, which belong to three types:

- benign;
- malignant;
- normal;

There are 208 normal, 63 benign and 51 malignant abnormal images.It also includes radiologist ′s truth-markings on the locations of any abnormalities that may be present. For each film, experienced radiologists give the type, location, scale, and other useful information of them. According to these experts ′ descriptions, the database consists of four different kinds of abnormities explained in 4.3

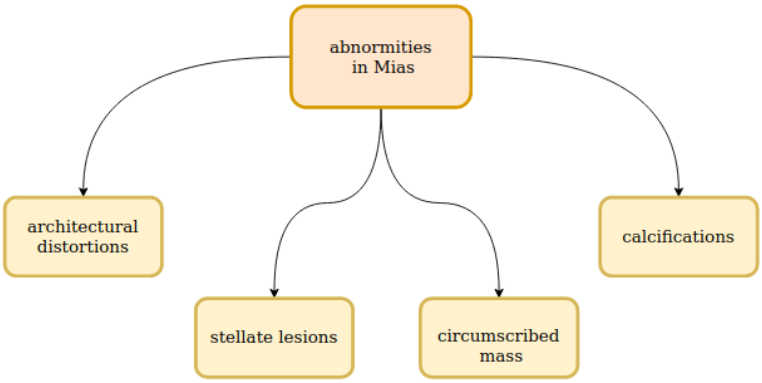

Figure 4.3. Content of intorduction file

The database possesses an introduction file, which includes some informations as reported in 4.4:

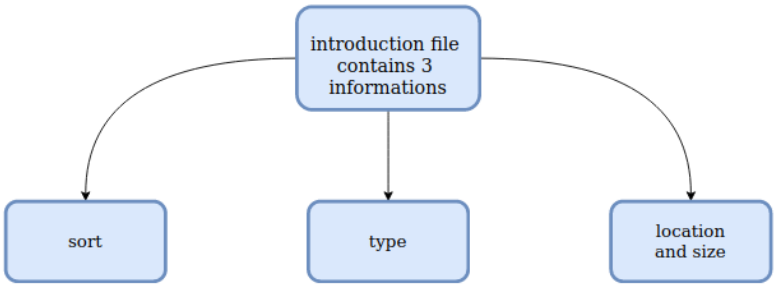

Figure 4.4. Content of intorduction file

type indicates which kinds mentioned above the abnormities belong to. **sort** indicates whether the abnormities are cancer or benign ones; **location and size** indicates the original coordinates and diameters of the abnormities.

4.1.3 INbreast

The INbreast database (Le et al., 2019) contains images acquired using MammoNovation Siemens full-field digital mammography and saved in the DICOM format. This dataset has a total of 410 images obtained from 115 cases and comprising 116 masses present in 107 images, full field digital mammography (FFDM) images with mass findings. These images are available in DICOM format and are of size 3328

* 4084 (or) 2560 * 3328 with 14-bit resolution. The boundary pixels of the mass are manually annotated and provided in XML format. Because of the smaller size of the dataset, a 5-fold cross-validation is performed on the dataset to gauge the model's performance. Additionally, this database includes data on breast density and BI-RADS assessment. The set includes examples of both normal mammograms and those with abnormalities. The abnormalities in the InBreast dataset includes masses, calcifications, asymmetries and architectural distortions.

In these two databases mammograms are taken from two angles:

- the mediolateral oblique (MLO);
- Cranial-Caudal (CC);

The MLO angle is an oblique view that includes the breast and armpit area âĂŹs upper outer quadrant. On the contrary to the MLO view, the CC angle depicts the entire breast area as it is a top-down view taken from above.

4.1.4 BreakHis

The **Breast Cancer Histopathological Image Classification (BreakHis)** is composed of 9,109 microscopic images of breast tumor tissue collected from 82 patients using different magnifying factors:

- 40X;
- 100X;
- 200X;
- 400X;

To date, it contains 2,480 benign and 5,429 malignant samples (700X460 pixels, 3-channel RGB, 8-bit depth in each channel, PNG format). This database has been built in collaboration with the P&D Laboratory Pathological Anatomy and Cytopathology, Parana, Brazil.
The dataset **BreaKHis** is divided into two main groups:

- benign tumors;
- malignant tumors;

Histologically benign is a term referring to a lesion that does not match any criteria of malignancy like *marked cellular atypia*, *mitosis*, *disruption of basement membranes*, *metastasize*. Normally, benign tumors are relatively innocents, presents slow growing and remains localized. Malignant tumor is a synonym for cancer: lesion can invade and destroy adjacent structures (locally invasive) and spread to distant sites (**metastasize**) to cause death. Samples present in dataset were collected by SOB method, also named partial mastectomy or excisional biopsy. This type of procedure, compared to any methods of needle biopsy, removes the larger size of tissue sample and is done in a hospital with general anesthetic. Both breast tumors benign and

malignant can be sorted into different types based on the way the tumoral cells look under the microscope. Various types/subtypes of breast tumors can have different prognoses and treatment implications. The dataset currently contains four histological distinct types of benign breast tumors:

- adenosis (A);
- fibroadenoma (F);
- phyllodes tumor (PT);
- tubular adenona (TA);

On the other hand four malignant tumors (breast cancer):

- carcinoma (DC);
- lobular carcinoma (LC);
- mucinous carcinoma (MC);
- papillary carcinoma (PC);

4.1.5 CSAW-S

CSAW-S is a dataset of mammography images which includes expert annotations of tumors and non-expert annotations of breast anatomy and artifacts in the image. It particular as described in (Matsoukas et al., 2020), it is a companion subset of CSAW, a large group of mammography data gathered from the entire population of Stockholm invited for screening between 2008 and 2015.
This dataset contains mammography screenings from 172 different patients with annotations for semantic segmentation. The patients are split into a test set of 26 images from 23 patients and training/validation set containing 312 images from 150 patients. The training/validation images are accompanied by cancer annotations by an expert radiologist and the testimages come with cancer annotations from two additional radiologists. Complementary labels are provided for the entire dataset in the form of full pixel-wise masks of each image corresponding to 11 additional highly imbalanced classes representing breast anatomy and other object.

Due to number of informations and flexibility of data, many of the dataset illustrated above were discarded. The two datasets that were appropriate for this work are:

- CBIS-DDSM;
- INbreast;

4.2 Scripts

This section briefly focuses on the scripts produced in order to work on the dataset chosen, prepare it and feed it to networks. The choice fell on CBIS-DDSM since it seemed to be the most complete and it offers a lot of images from many patients. Moreover a great advantage of this dataset is that it is bilanced, so equal number of images for each label.
The most important scripts are listed below:

- Conversion_dataset.py;
- MassTrainingGenpkl.py;
- convert.sh;
- simplify.sh;
- data_handling.py;
- labelstraining.py;

Conversion_dataset.py is a script used to convert the dataset composed of DICOM files into a png one as illustrated in the scheme in 4.5:

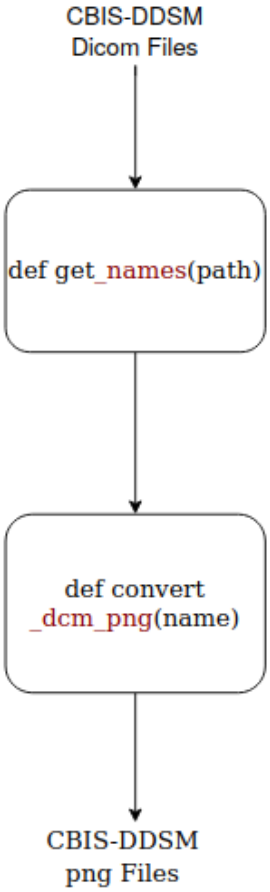

Figure 4.5. a simple scheme that explain the aim of **Conversion_dataset.py**

The second script of the list, **MassTrainingGenpkl.py**, is a very **important** file. In fact it generates a binary file called **exam_list_before_cropping.pkl** that is fondamental for the **cropping phase** since **crop_mammogram** takes as input this file.pkl. Next, **convert.sh** and **simplify.sh** are very short but important scripts in *bash*, that modify the structure of the folders in CBIS-DDSM. It is possible to see the original structure of the dataset before the execution of **convert.sh** and **simplify.sh** in 8.1

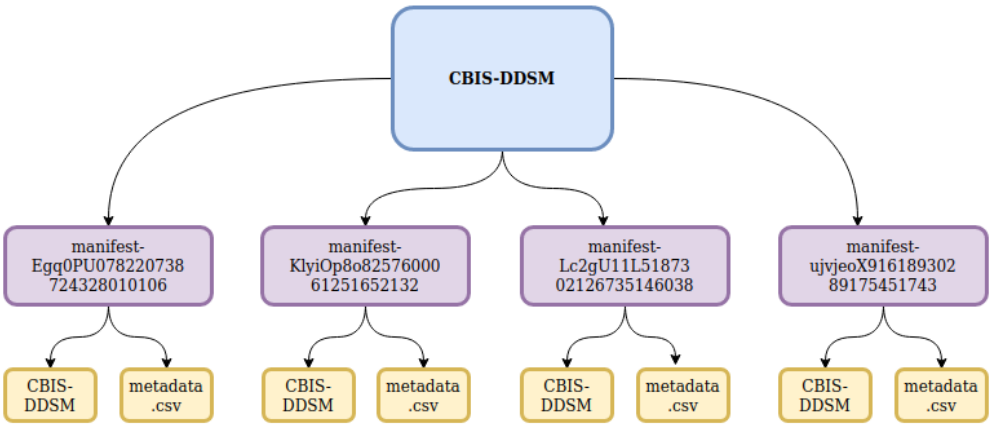

Figure 4.6. first part of the structure of CBIS-DDSM

The new structure of the dataset after **convert.sh** and and **simplify.sh** is reported in 4.7

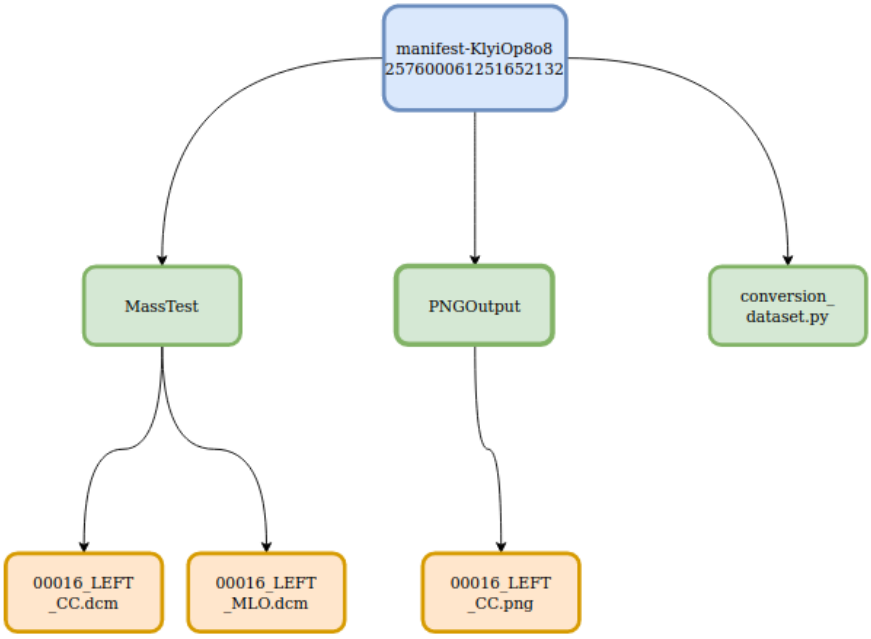

Figure 4.7. this structure shows the depth of every dcm file before applying any scripts

labelstraining.py is fondamental to performe the training. In fact the utility fuction**datasets.ImageFolder** expects both the *training* and the *test* to be structured as illustrated in 4.8:

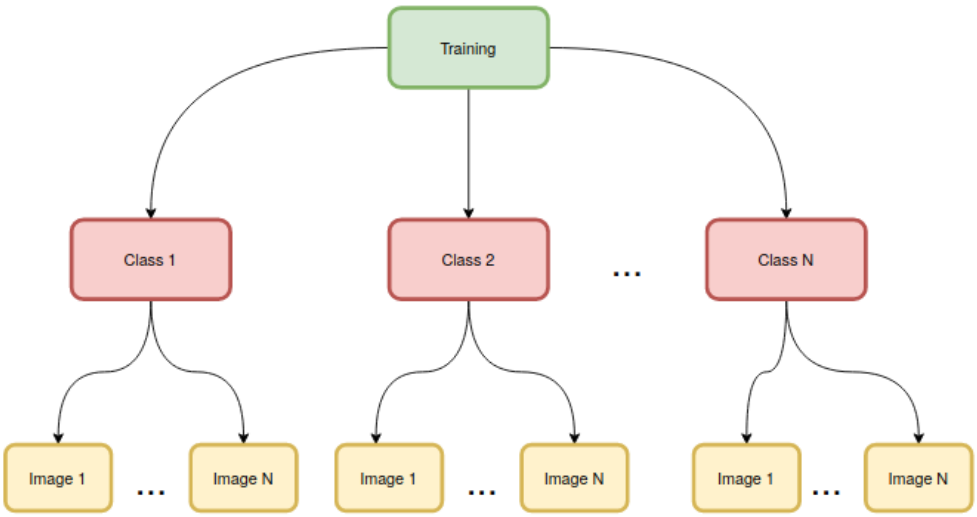

Figure 4.8. training general structure

Training and **test** have to be divided into the their rispective folders according to the labels of the dataset. This is in few words what the script **labelstraining.py** achives.
The new structure obtained is represented in 4.9:

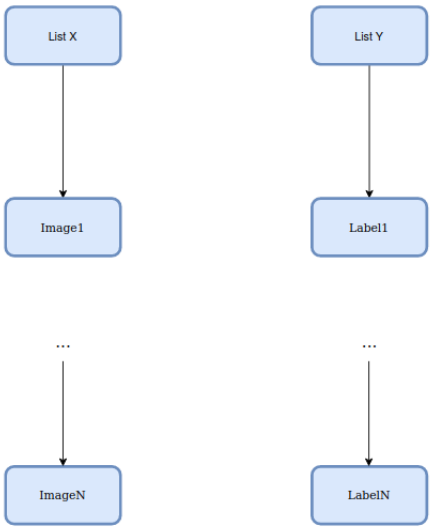

Figure 4.9. structure of *list x* and **list y**

Next just a training in pytorch.
A huge problem to solve was that the dataset is very big so it is the memory required. For this reason on **google colab** some solutions have been adopted.
The first has been to install **imagemagick** and then reduce of about the half of the original dimensions. As it is possible to see the many of these commands are commented. The last one computes all the command below in one.

Chapter 5

Image classification on CBIS-DDSM: plots and results

The first approach of the project consist in **Image Classification** on CBIS-DDSM, between *benignant* and *malignant* tumors. This chapter illustrates the plots obteined after having trained the models on CBIS-DDSM Mass. For this chapter six popular achitectures have been evaluated, (ResNet-18, AlexNet, DenseNet-121, Inception_V3, GoogleNet and ShuffleNet) they are widely used CNN architectures and they are represented in 5.1 as follows:

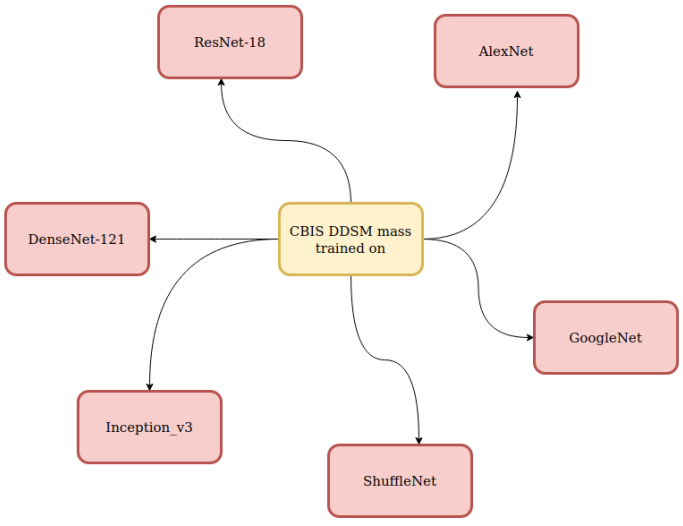

Figure 5.1. models where CBIS-DDSM mass part was tested

5.1 ResNet

For this model three plots are avilable and they are shown below.

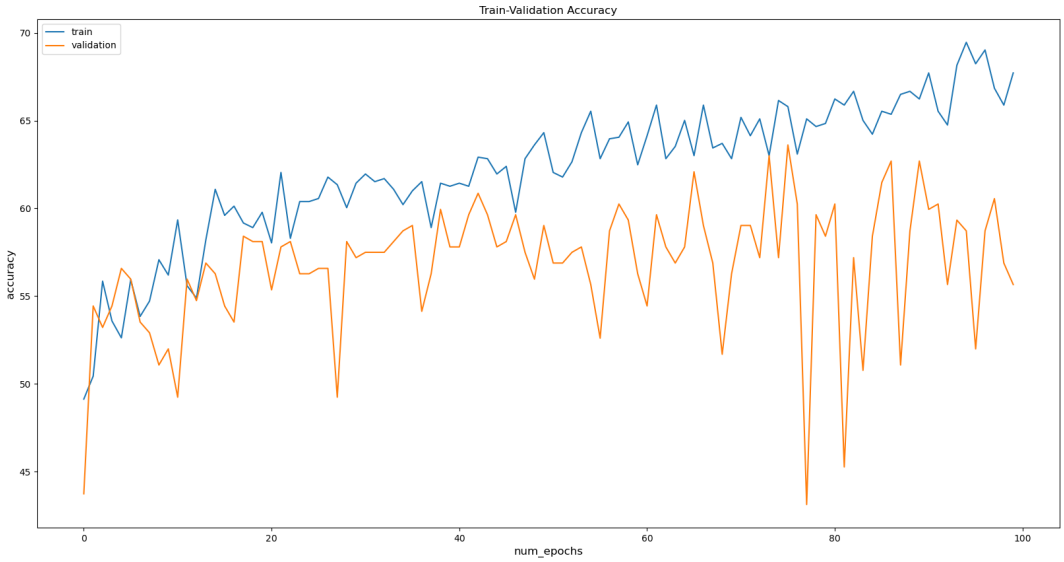

Figure 5.2. ResNet-18 trained on whole Mass on 100 epochs

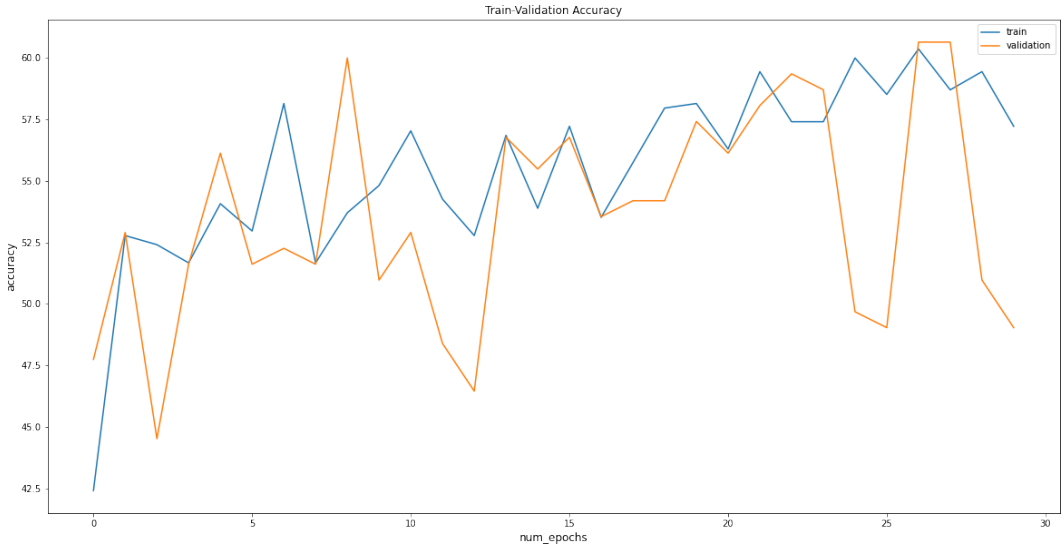

Figure 5.3. ResNet-18 trained on CC Mass on 30 epochs

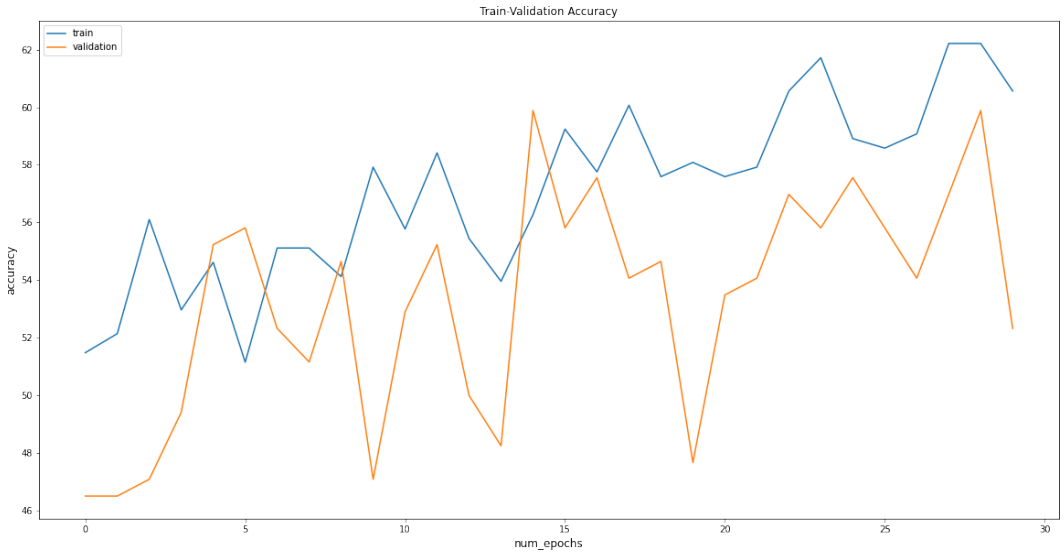

Figure 5.4. ResNet-18 trained on MLO Mass on 30 epochs

The first plot 5.2 has been obtained training the model for 100 epochs on the whole Mass. It is possible to see that there is *overfitting* by looking at the increasing difference between the blue and orange lines. In particular the former represents the *training evolution* over time and the latter refers to *validation*. It is important to remember that in general it would be intelligent to consider another metric, the **f1-score** but CBIS-DDSM is a balanced dataset, so number of tumors and non tumors cases are quite the same and this fact makes useless to consider this metric. The accuracy value reaches a result of 70.
The training has been executed on Google Colab with a *batch size* of 8, due to memory problems.

The second plot 5.3 has been obtained training the model for 30 epochs on the CC part of Mass. As it is possible to see the overfitting is reduced respect to the previous plot, however in some parts it is still present. The accuracy gains value of about 60. The training has been executed on Google Colab with a *batch size* of 8, due to memory problems.

The third plot 5.3 has been obtained training the model for 30 epochs on the MLO part of Mass this time. As it is possible to see there is more overfitting respect to 5.3. The accuracy gains value of about 62 so there is no big difference with the CC part.The training has been executed on Google Colab with a *batch size* of 8, due to memory problems.

5.2 AlexNet

For **AlexNet** only one plot is reported below, since the plots related to the 30 epochs on CC parts and MLO parts were stable and they did not perform well so it was appropriate not to mention them. Below 5.5 represents the plot trained on 100

epochs on the whole Mass part of CBIS-DDSM. As it is possible to see the model does not perform well, in fact there is a lot of overfitting, exept few points on the plot. In addition during the first 20 epochs the model seems not to learn at all and last, the train line does not increase as expected. In fact it goes from 52 to 56 of accuracy and the difference with **ResNet-18** is highlighted a lot. Comparing 5.4 and this plot it is possible to conclude that for this dataset ResNet-18 performes well, whereas AlexNet is in absolute not a good model for CBIS-DDSM.
The training has been executed on Google Colab with a *batch size* of 4, due to memory problems.

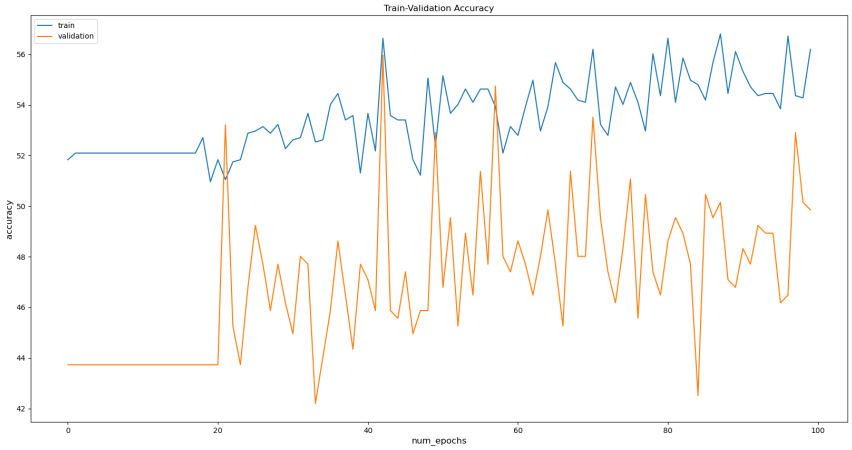

Figure 5.5. AlexNet trained on 100 epochs on CBIS-DDSM Mass part

5.3 DenseNet

For this model **DenseNet121** has been used and three different plots are available. They are reported below:

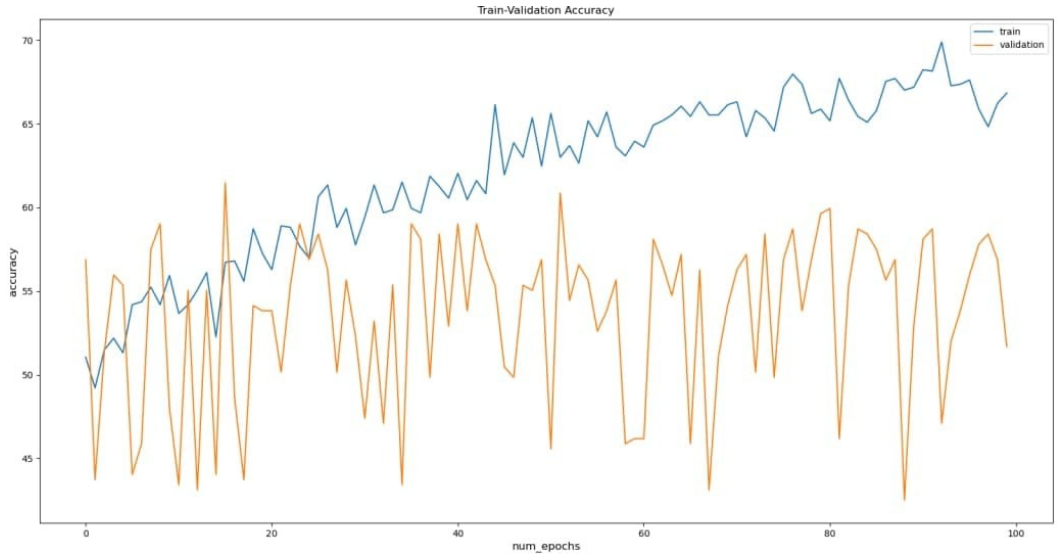

Figure 5.6. DenseNet-121 trained on whole Mass on 100 epochs

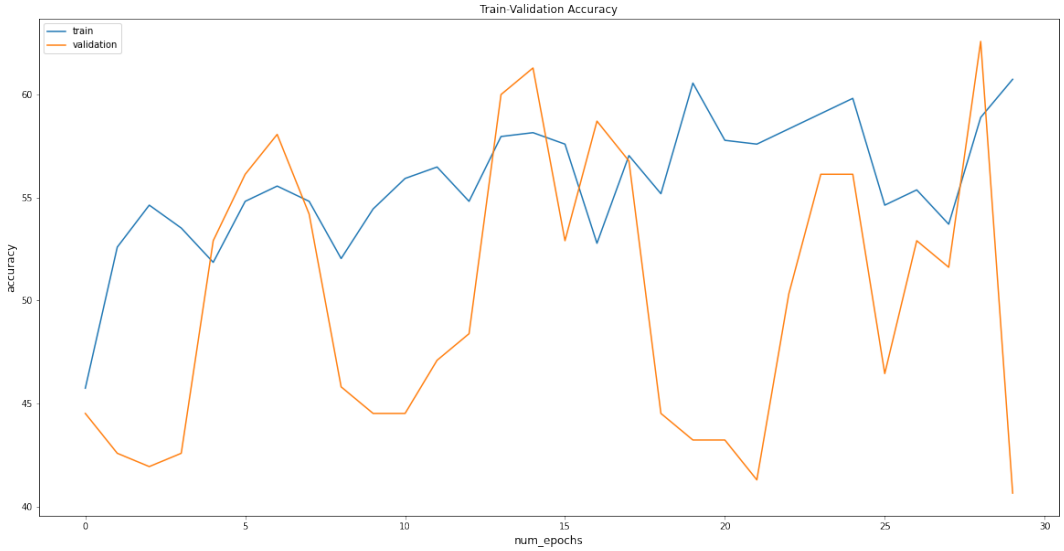

Figure 5.7. DenseNet-121 trained on CC Mass on 30 epochs

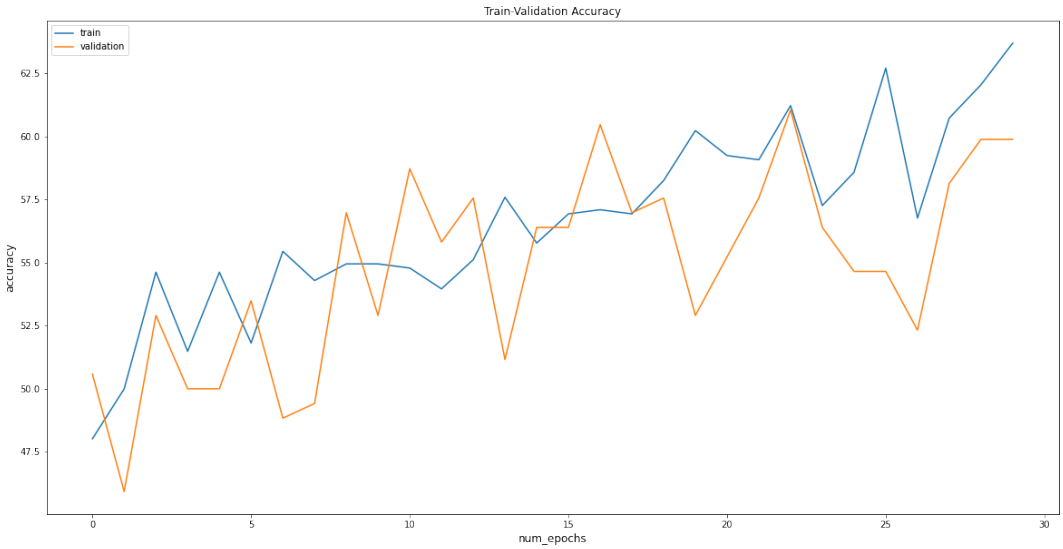

Figure 5.8. DenseNet-121 trained on MLO Mass on 30 epochs

5.6 represents the results obtained on a training of 100 epochs on the whole Mass part of CBIS-DDSM. This plot shows some good and bad characteristics. The good ones are that the train line, the blu one, increases a lot. It means the model is actually learning in fact it starts from 51 and finishes at 68, reaching some epochs before 100 the 70 value of accuracy. Moreover the train is way more stable than the validation, the yellow line, that shows many ups and downs. There is overfitting, in particular from the epoch 25, while in the first epochs there are many points of intersection beetween the two lines.
The training has been executed on Google Colab with a *batch size* of 8, due to memory problems.

5.7 is the second plot and it represents the results obtained training the model on the CC part of Mass, for 30 epochs. The plot highlight a huge instability, probably related to the *batch_size* value. In addition it is possible to notice how the train line increases from a value of 45 to a value of 60, but there is a lot of overfitting since the validation increase and decrease in a totally different way.

5.8 is the third plot and it represents the results obtained training the model on the MLO part of Mass, for 30 epochs. Respect to 5.7 it is more stable and it can be considered a good result. In fact both train and validation increase in a quite similar way. The train learns since it goes from 47 to 63 and for sure the overfitting is strongly reduced wit respect to the previous plot.

5.4 GoogleNet

For this model three plots are reported below:

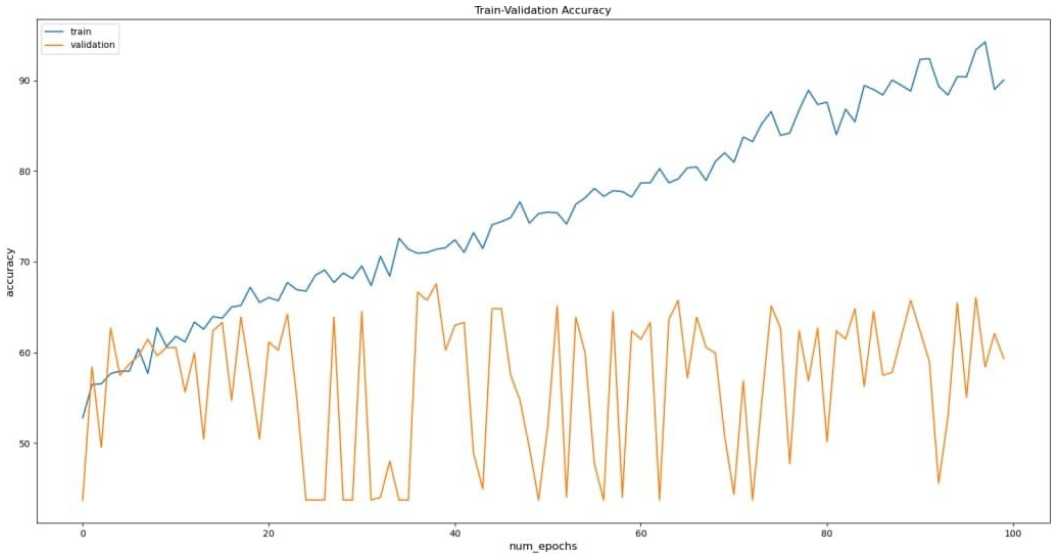

Figure 5.9. GoogleNet trained on whole Mass on 100 epochs

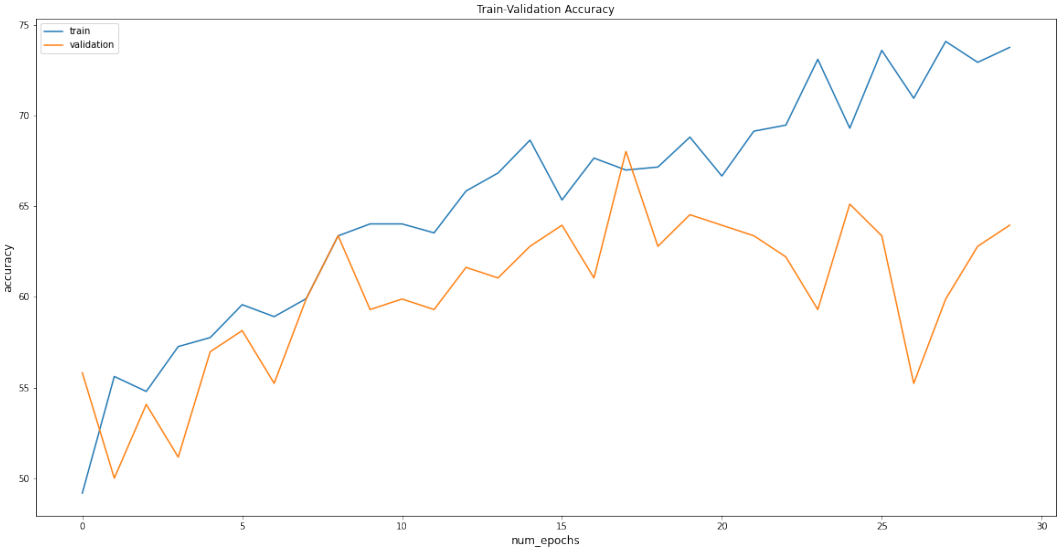

Figure 5.10. GoogleNet trained on MLO Mass on 30 epochs

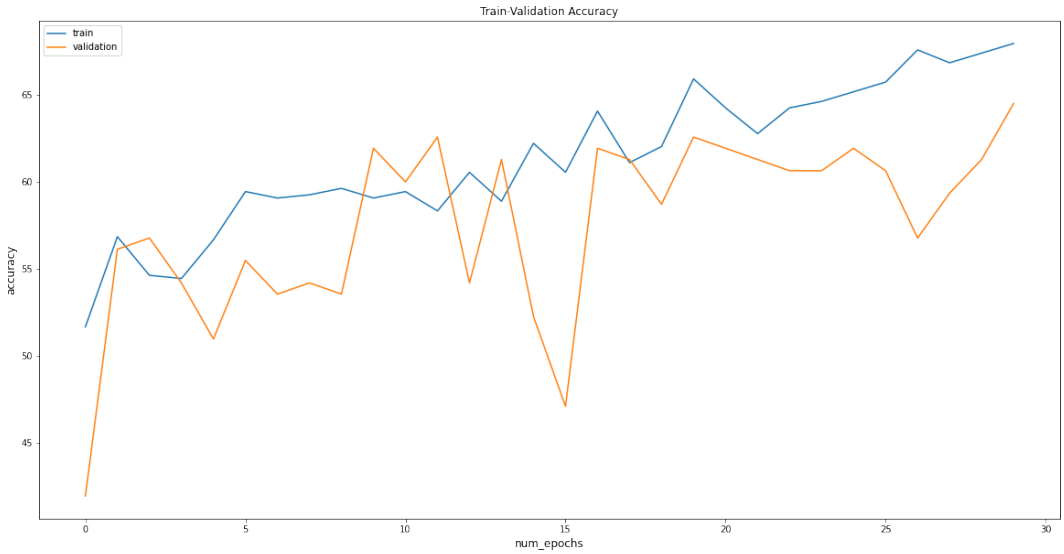

Figure 5.11. GoogleNet trained on CC Mass on 30 epochs

The first plot, 5.9 represents the performance of the model trained for 100 epochs on the mass part of CBIS-DDSM. The train line increases a lot, from a value to 52 to 90. However the overfitting is a lot since the validation line the yellow one does not increase a lot. Moreover it remains stable on the same low values and this fact makes this result not good at all, even if the train line was very good. In fact the validation does not perform well. The training has been executed on Google Colab with a batch_size of 8.

The second plot, 5.10 is the plot related to the model trained for 30 epochs of the MLO split. In this plot the overfitting is strongly reduced, in fact both train e validation increase in a similar way. However the overfitting increases a little during the last 10 epochs as it is possible to see. The train line reaches an accuracy value of 75, starting from less than 50, it means the model is learning a lot. It makes this plot very good.

The third plot, 5.11 is the plot related to the model trained for 30 epochs of the CC split. At a first sight it does not appear good, and it is very different respect to 5.10 from a point of view of permormances. The train line goes from a value of 50 to something more of 65, so it is reduced respect to 5.10. A good characteristic of this plot is that the overfitting is not a lot, exept from some points. In general 5.11 can be considered good but 5.10 is better.

5.5 Inception

The plots obtained for this model are three and they are reported below. Until now, for the other models the first plot was related to the training of the whole mass part of **CBIS-DDSM** on 100 epochs, whereas the second and the third were the splits related to CC and MLO, trained on 30 epochs. For **inception_v3** the situation is a

bit different, since the first plot was training only on 20 epochs because it seemed not to learn at all and also because of the great overfitting from epoch 30. It is known that to reduce overfitting two solutions can be adopted, one is data augmentation but in this case it did not healp at all and the other is to reduce the number of epochs.

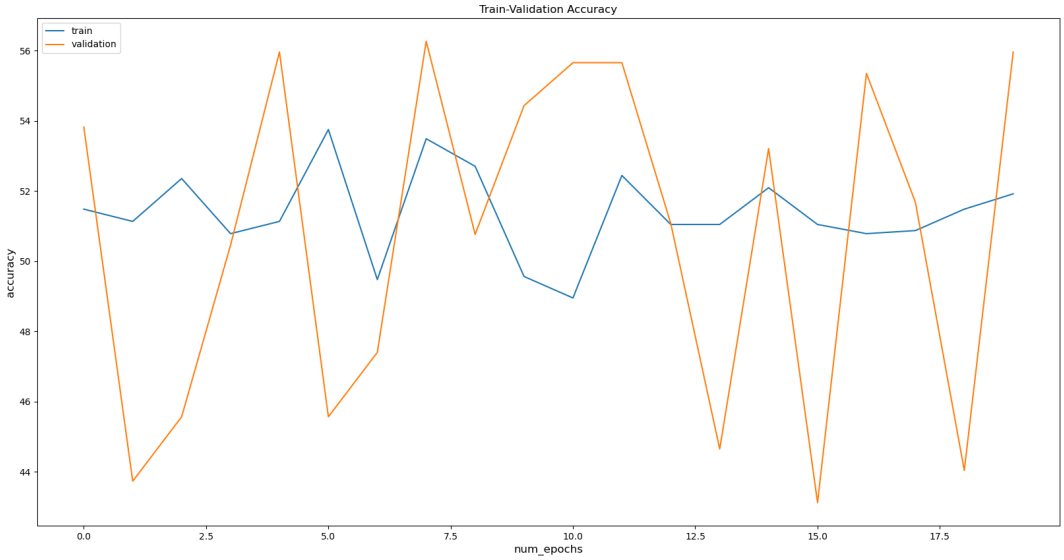

Figure 5.12. Inception trained on whole Mass on 20 epochs

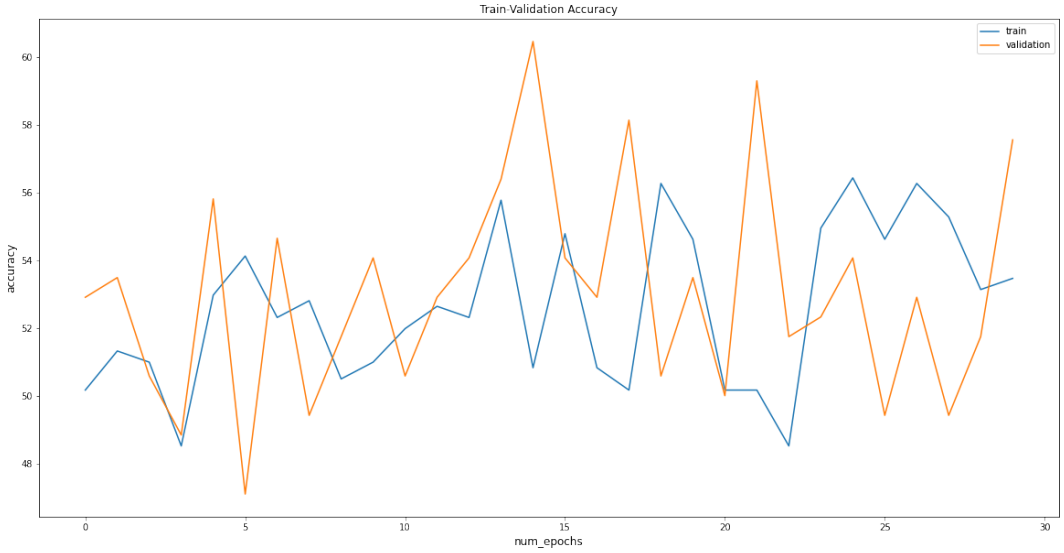

Figure 5.13. Inception trained on MLO Mass on 30 epochs

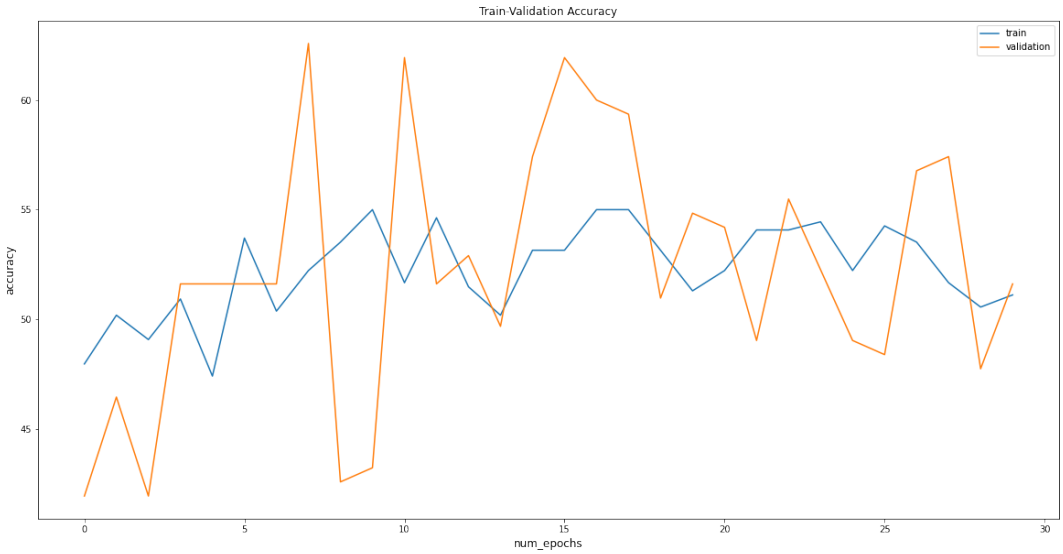

Figure 5.14. Inception trained on CC Mass on 30 epochs

The first plot, 5.12 represent the training on the whole mass part and it can be considered bad from any point of view. In fact the train line remains quite stable but also the validation seems to learn nothing. Moreover the validation is very unstable. The training has been executed on Google Colab with a batch size of 8, due to memory problems.

The second plot, 5.13 in very bad too. The train line acts the same as 5.12 starting from 52 and remain on this value for the whole training. It is neither appropriate to comment the overfitting since for this model the main problem is that it does not fit the classification problem analized in this project. In few words also 5.13 that take into consideration only the MLO part is not good. The training has been executed on Google Colab with a batch size of 8, due to memory problems.

The third plot, 5.14 confirms the description of the previous plots. The train remains stable, the validation is not costant and does not learn. The training has been executed on Google Colab with a batch size of 8, due to memory problems. In few words it is possible to assert that *inception_v3* is not an appropriate model to detect breast cancer.

5.6 ShuffleNet

For this model three plots are reported below:

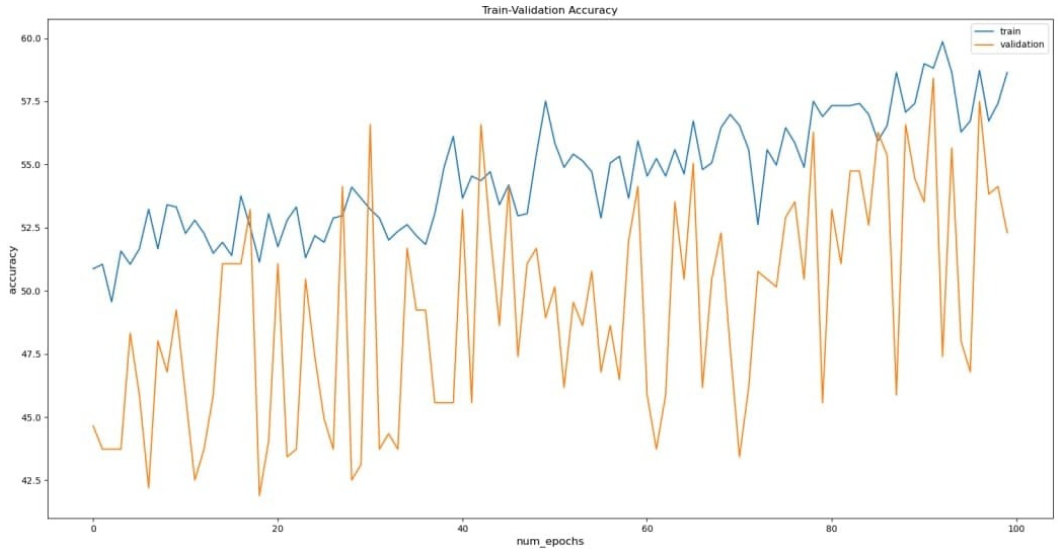

Figure 5.15. ShuffleNet trained on whole Mass on 100 epochs

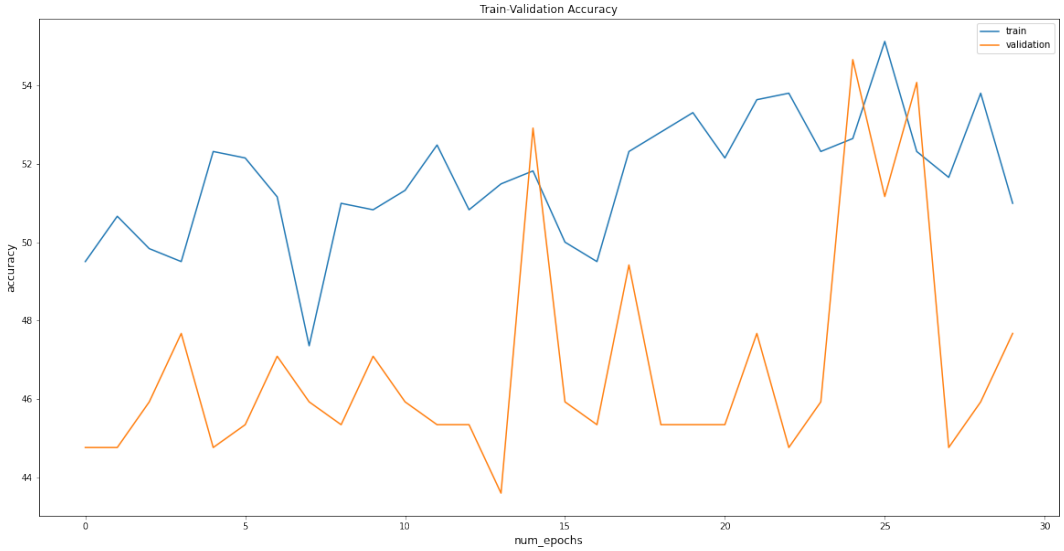

Figure 5.16. ShuffleNet trained on MLO Mass on 30 epochs

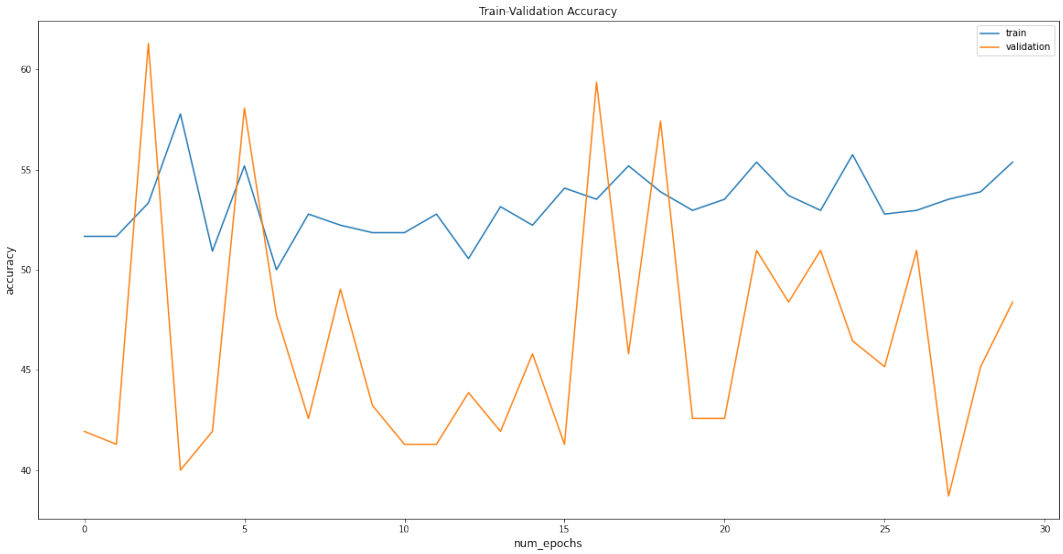

Figure 5.17. ShuffleNet trained on CC Mass on 30 epochs

5.15 is the first plot and it represents the model trained on the whole Mass split of CBIS-DDSM on 100 epochs. The first characteristic that is highlighted is the instability. Both train and validation shows instability but in particular the validation. This can be related to the small value of the *batch_size*, in fact the model has been trained with a *batch_size* of 4 again for memory issues. Moreover the train line goes from 50 to 60 so there is acknowledge but not so much. The overfitting, exept in few points, is a lot. In general the results are not good.

The second plot, 5.16, represents the model trained on the MLO part of Mass split of CBIS-DDSM on 30 epochs. This plot has two important problems, on one hand the overfitting is a lot since the blue and the yellow line are , on the other the model does not learn at all since the train line goes from 50 to about 54.

The third plot, 5.17, represents the model trained on the CC part of Mass split of CBIS-DDSM on 30 epochs. This plot is even worse than 5.16 since the train line remains quite costant and exept some points there is a lot of overfitting.

Chapter 6

Image classification on CBIS-DDSM, ROIs

6.1 ROI Extraction Methods in CBIS

ROIs are binary mask images delineating the abnormality in mammograms. For this reason ROIs represent important information in the field of breast cancer, since it is possible to reduce the region of analisis, focusing just on these portions of mammograms.
ROIs are provided in CBIS-DDSM. The masks were used to define a square which completely enclosed the ROI that means *region of interest*. As already exolained they represent important addition information mainly with the aim to increase the size of the training data. In theory better results are expected with these experiments with respect to the previous ones, image classification on the whole images.

6.2 Cropping Roi

Since ROIs appear as masks, where the relevant part is represented in white and the irrelevant part is represented in black, before being used as input for the models, ROIs absolutely need to be preprocessed and prepared. For this reason a script in python has been created, **scriptROI.py**. The code is reported and explained in the following lines.

```
import os
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
from PIL import Image
import PIL

path_roi = '/media/federica/Loserino/CBIS-DDSM/CalcTrainingROI/PNGOutput/'
path_img = '/media/federica/Loserino/CBIS-DDSM/CalcTrain/PNGOutput/'

path_out = '/media/federica/Loserino/CBIS-DDSM/path_out/'
box_size = 600

def bbox1(img):
    a = np.where(img != 0)
    bbox = np.min(a[0]), np.max(a[0]), np.min(a[1]), np.max(a[1])
    return bbox
```

Figure 6.1. first part of *scriptROI.py*

The first important thing in this script as shown in 6.2 is defining the two folders where ROIs and mammograms are stored as it is possible to see. In fact for each ROI is associated to a mammogram, so each mask corresponds to an image. This is the role of path_roi and *path_img*. *path_out* is the new folder in which the new images will be stored.
Thereafter, the function **bbox1** is defined. It works only on ROIs masks and it considers exclusively the regions with pixel equal to 1. This is why the white region, that is the important part of the image, is associated to pixels with value 1, the black one that is the irrelevant part of the image, is associated to pixels with value 0. Later, four points are considered since the aim of the function is to return a rectangular so a bounding box. These four points are:

- min a[0];
- max a[0];
- min a[1];
- max a[1];

Their representation in the Eucledian space appears as follows and it is possible to see how the result is a rectangular.
Moreover *box_size* is a variable set to store the ideal dimension of each bounding box.

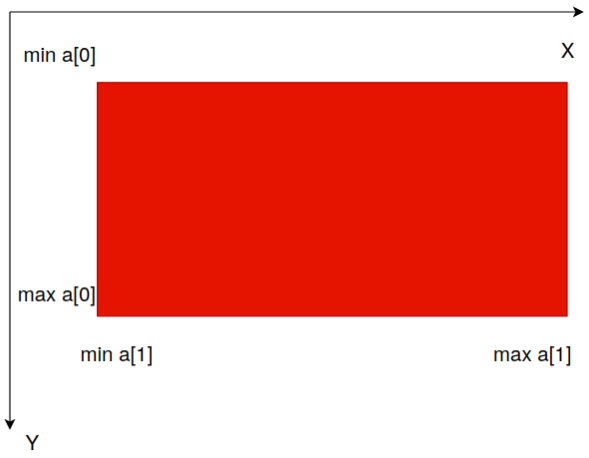

Figure 6.2. result of bbox1 function

The script continues as illustrated in 6.3 with a **for loop** and next it reads the content of both folders *roi* and *img*. Then a simple *if control* checks that the shape of each tuple is the same, since this is a fondamental requirment for the whole idea. Now the function **bbox1** is called on each roi and returns four vertices named my, My, mx, Mx. They are very important since the next step consists in calculating the center on each axis, *cy* and *cx*.

```
for f in os.listdir(path_roi):
    try:
        roi = plt.imread(path_roi + f)
        img = plt.imread(path_img + f)
    except FileNotFoundError:
        continue
    if roi.shape != img.shape:
        continue
    my, My, mx, Mx = bbox1(roi)
```

Figure 6.3. second part of *scriptROI.py*

In 6.4 it is highlighted how the centers and the bounding boxes around the ROIs are calculated and as it has been already said the four vertices returned by the function *bbox1* are very important. In fact *cx* and *cy* corresponds to the centers on the two axis. Next, the bounding boxes are creating taking into account the variable *box_size*, whose meaning has been clarified before. Of course, since the center has been taking in consideration, when creating the bounding boxes it is necessary to divide *box_size* by two.

```
# trova centro
cy = (my + My) // 2
cx = (mx + Mx) // 2

# crea bbox
mx = cx - (box_size // 2)
Mx = cx + (box_size // 2)

my = cy - (box_size // 2)
My = cy + (box_size // 2)
```

Figure 6.4. third part of *scriptROI.py*

The last part of the script focuses on some important conditions faced in order to prevent all the special cases and issues when creating bounding boxes on different images. In 6.5 all these conditions are considered and controlled through some if statments.

```
if mx < 0:
    mx = 0
    Mx = box_size

if Mx > img.shape[1]:
    mx = img.shape[1] - box_size
    Mx = img.shape[1]

if my < 0:
    my = 0
    My = box_size

if My > img.shape[0]:
    my = img.shape[0] - box_size
    My = img.shape[0]
crop = img[my:My, mx:Mx]
print(f)
plt.imsave(path_out + f, crop, cmap='gray')
```

Figure 6.5. last part of scriptROI.py

Now that ROIs are preprocessed it is possible to train them with some models. The choice fell on those models that in the previous chapter performed better. They are ResNet-18, DenseNet-121 and GoogleNet as 6.6 shows.

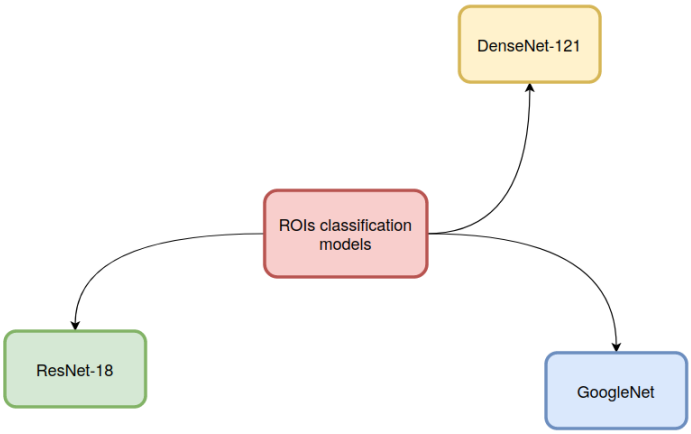

Figure 6.6. Models chose for ROIs classification

Plots are not reported since this part highlighted some problematics. In fact all the architectures in 6.6 seemed not to work at all with the ROIs dataset. For this reason some solution to face the problem needed to be found. Two ways to approach the problematic have been adopted:

- Explainable AI;
- Radiologists ′ classification;

6.3 Explainable AI

Explainable AI is one of the most useful tools in the field of Machine Learning. Explainable AI is also known as XAI and it is an emerging field in machine learning that aims to address how black box decisions of AI systems are made. This area inspects and tries to understand the steps and models involved in making decisions. In fact machine learning models are not always easy to understand as in the case of this project so the necessity to interpret them can rise. *Explainable AI* can be thought as a way to understand the errors a model makes and also the reasoning behind its predictions. In few words this technique can be defined as a group of methods and techniques in the application of artificial intelligence technology such that the results of the solution can be understood by human experts.

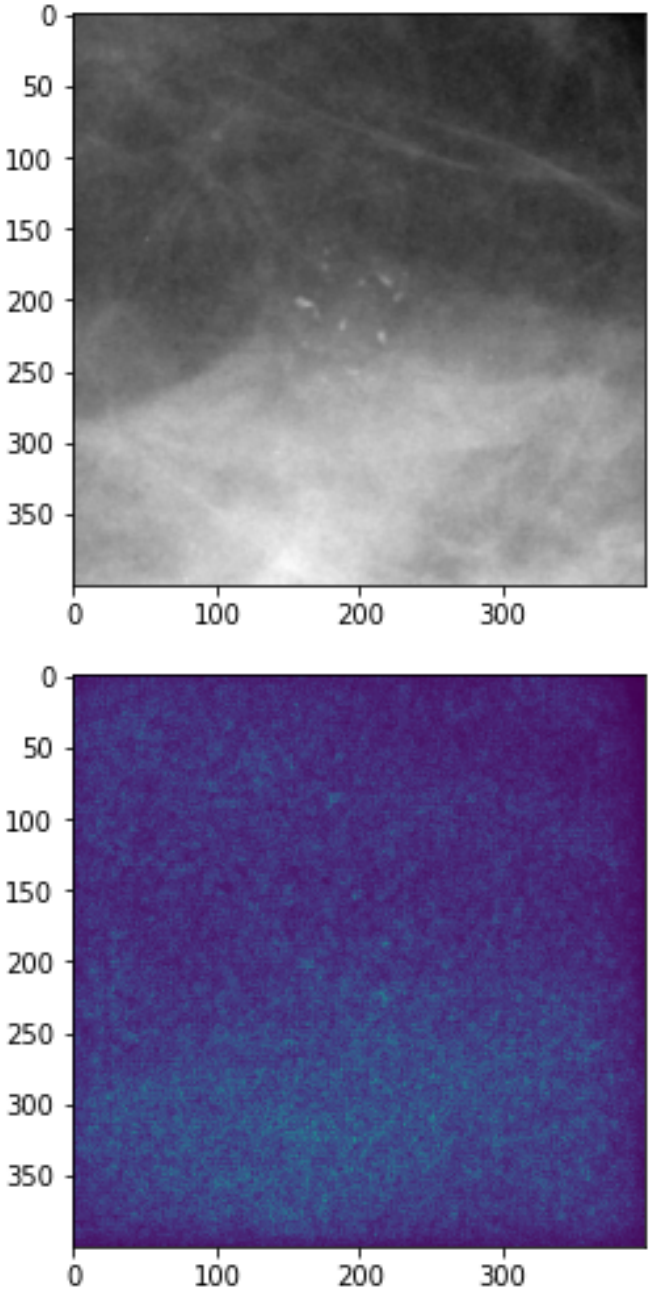

Figure 6.7. Explainable AI performed on ResNet-18 with ROIs

6.4 Radiologists ′ classification

After having trained the ROIs dataset extracted from CBIS-DDSM, it has been clear that the results were not good at all if considering a classification problem. In fact the validation did not manage to overcome the value of about 55, no matter if the network was ResNet, DenseNet or AlexNet. All the models seemed to performe in the same, bad way. It seemed that any network could not learn at all using

this dataset. Before thinking about a solution to obtain better performances, an important test has been done. In fact the reasons why *CBIS-DDSM* ROIs did not perform well can be related to the use of models that are not appropriate for this problem, or it has to be reconducted to the composition of the dataset. To answer to these doubts five specialists have examinated 20 images from ROIs of *CBIS-DDSM* and have classified them as benign or malign adding some comments on how they managed to state if a mammogram is benign or not.
The images chosen to be examinated are reported in the following table:

Calc Benign	Calc Malign	Mass Benign	Mass Malign
007 left cc	005 right mlo	001 left cc	004 left cc
011 left cc	012 left cc	023 right cc	027 right cc
013 right mlo	016 left cc	039 right mlo	044 right cc
019 right cc	0.20 left cc	057 right cc	090 left cc
022 left cc	0.34 right cc	059 left mlo	094 right cc

It is possible to see in the table that the 20 images are extracted both from calc and mass, balancing the two labels. So, in a teorethical way, networks and radiologists should state that 10 images are benign mammograms, whereas the remaining 10 are malign. It as been explained that networks did not perform well, however also human predictions are not better as it will be explained later, since a mean of 11/20 images were given the right label.
Five specialists took part and as a consequences five tables follows.

6.4.1 first analisis: dott.ssa Angela Stasio

Mammogram	Radiologist's prediction	Actual prediction	Radiologist's comments
1	malign	malign	lump with jagged edges
2	non definibile	benign	
3	malign	malign	clusters of small and grouped microcalcifications
4	benign	benign	coarse and deposited calcifications along a duct
5	malign	benign	clusters of small and grouped microcalcifications
6	malign	malign	jagged thickening with calcium deposits
7	benign	benign	coarse, non-clustered calcifications
8	malign	malign	irregular thickening with calcifications
9	benign	benign	scattered calcifications
10	benign	malign	scattered and coarse calcifications
11	undefined	benign	
12	malign	malign	irregular lump
13	malign	benign	multiple densification
14	malign	malign	jagged thickening
15	malign	malign	multiple irregular nodules
16	benign	benign	regularity
17	malign	malign	irregular lump
18	benign	malign	lump with irregular margins
19	malign	benign	irregular thickening
20	undefined	benign	dense udder

The first information represented in the table above is provided by dott.ssa Angela Stasio, UOSD Chirurgia senologica e ricostruttiva from Sandro Pertini Hospital, in Rome. After analizing the informations provided by dott.ssa Stasio it is possible to see that her predictions fit the original labels only for 12/20 images. It is important to notice that dott.ssa Stasio provided also many comments on how she stated her predictions and in fact there are correlations between each image and its comments. It is also important to remember that dott.ssa Stasio is not a radiologist, whereas the other four doctors are. It is a great opportunity for this analysis since many points of view from different experts of the field of breast cancer are offered.

6.4.2 second analisis: dott.ssa Stefania Minuto

Mammogram	Radiologist's prediction	Actual prediction	Radiologist's comments
1	malign	malign	lump with irregular margins
2	benign	benign	absence of nodular images, microcalcifications and pathological distortions
3	benign	malign	microcalcifications with regular, low density and grouped morphology
4	benign	benign	ductal microcalcifications
5	malign	benign	cluster of irregular microcalcifications on parenchymal distortion
6	malign	malign	irregular parenchymal distortion with calcified deposits
7	benign	benign	regular calcifications on sharply contoured opacity (likely cyst)
8	malign	malign	cluster of polymorphic microcalcifications on irregular densification
9	undefined	benign	underexposed, blurred image
10	benign	malign	coarse calcifications
11	undefined	benign	blurred image perhaps presence of artifacts
12	malign	malign	central nodule with spiculate margins
13	malign	benign	multiple nodular thickening and parenchymal distortions
14	malign	malign	polymorphic microcalcifications on irregular densification
15	malign	malign	multiple nodular thickenings and enlarged axillary lymph nodes
16	malign	benign	retroareolar glandular distortion and multiple nodules
17	malign	malign	nodules with slightly regular outlines
18	benign	malign	nodular opacities with regular outlines
19	benign	benign	irregular thickening
20	undefined	benign	technical error

The second analysis has been an accurate work done by the dott.ssa Stefania Minuto, (Dirigente medico UOC Diagnostica per Immagini radiologia) radiologist from Hospital Sandro Pertini in Rome. It is important to notice that despite the fact that in this case the comments associated to each mammogram are more specific and own of the field of radiologists, the statments related to each image are quite the same stated from dott.ssa Angela Stasio. This can be considered a good results since the human detection does not only overcome the performances of neural networks but seems to have feedbacks between different experts. This could be related to some difficulties own of CBIS-DDSM. In this case the right predictions of the 20 images for test was 11/20.

6.4.3 third analisis: dott.ssa Raffaella Poggi

Mammogram	Radiologist's prediction	Actual prediction	Radiologist's comments
1	malign	malign	lump with irregular margins
2	benign	benign	absence of pathological findings
3	benign	malign	regular microcalcifications
4	benign	benign	regular microcalcifications
5	malign	benign	irregular microcalcifications upon thickening
6	malign	malign	distortion with some microcalcification
7	benign	benign	sharp edge opacity with peripheral microcalcifications
8	malign	malign	thickening at irregular margins with microcalcifications
9	undefined	benign	underexposed image with poor compression
10	benign	malign	gross calcifications apparently dystrophic
11	undefined	benign	images with artifacts
12	malign	malign	spiculated image to QQEE; there is also a benign nodule on QQII
13	malign	benign	nuanced nodule
14	malign	malign	thickening and microcalcifications with crushed stone
15	malign	malign	multiple poorly definable nodules and lymphadenopathies
16	malign	benign	multiple nodules and retroareolar distortion
17	malign	malign	small QQINT nodule with irregular contours
18	benign	malign	regular lump
19	benign	benign	absence of pathological findings
20	undefined	benign	underexposed and slightly drawn image

The third analysis has been an accurate work done by the dott.ssa Raffaella Poggi, (Dirigente medico UOC Diagnostica per Immagini radiologia) radiologist from Hospital Sandro Pertini in Rome. In this case the right predictions of the 20 images for test was 11/20. Dott.ssa Raffaella Poggi as her colleagues offers an accurate description on how she steted her results. In fact analizing the fourth column of the table it is possible to see similarities with respect to what the mammogram shows. Moreover there is an agreement with what stated by the other esperts.

6.4.4 fourth analisis: dott.ssa Antonietta Mazzone

Mammogram	Radiologist's prediction	Actual prediction	Radiologist's comments
1	malign	malign	lump with irregular margins
2	benign	benign	absence of nodular images, microcalcifications and pathological distortions
3	benign	malign	microcalcifications with regular low density and grouped morphology
4	benign	benign	ductal microcalcifications
5	malign	benign	cluster of irregular microcalcifications upon thickening
6	malign	malign	distortion with some microcalcification
7	benign	benign	sharp edge opacity with peripheral microcalcifications
8	malign	malign	thickening at irregular margins with microcalcifications
9	undefined	benign	underexposed, blurred image
10	benign	malign	coarse calcifications
11	undefined	benign	images with artifacts
12	malign	malign	central nodule with spiculate margins
13	malign	benign	nuanced nodule
14	malign	malign	thickening and microcalcifications with crushed stone
15	malign	malign	multiple nodular thickenings and enlarged axillary lymph nodes
16	malign	benign	retroareolar glandular distortion and multiple nodules
17	malign	malign	slightly irregular contoured nodules
18	benign	malign	nodular opacities with regular outlines
19	benign	benign	absence of pathological findings
20	undefined	benign	technical error

This table shows another important and accurated work directed by dott.ssa Antonietta Mazzone, (Dirigente medico UOC Diagnostica per Immagini radiologia) radiologist from Hospital Sandro Pertini in Rome. The interesting aspect of this third analysis is that there are similarities with the previous ones. The hypotesis that the problem could be related to the composition of the dataset arises. Maybe the dataset is not appropriate for a classification problem, when refering to the ROIs. Dott.ssa Mazzone stated 11/20 mammograms as expected by CBIS-DDSM, and her annotations are similar as her Dott.ssa Poggi.

6.4.5 fifth analisis: team of expercts from Hospital Sandro Pertini, Rome

Mammogram	Radiologist's prediction	Actual prediction	Radiologist's comments
1	malign	malign	lump with irregular margins
2	benign	benign	absence of pathological findings
3	benign	malign	regular microcalcifications
4	benign	benign	regular microcalcifications
5	malign	benign	irregular microcalcifications upon thickening
6	malign	malign	distortion with some microcalcification
7	benign	benign	sharp edge opacity with peripheral microcalcifications
8	malign	malign	thickening at irregular margins with microcalcifications
9	undefined	benign	underexposed image with poor compression
10	benign	malign	gross calcifications apparently dystrophic
11	undefined	benign	images with artifacts
12	malign	malign	spiculated image to QQEE; there is also a benign nodule on QQII
13	malign	benign	multiple nodular thickening and parenchymal distortions
14	malign	malign	polymorphic microcalcifications on irregular thickening
15	malign	malign	multiple poorly definable nodules and lymphadenopathies
16	malign	benign	multiple nodules and retroareolar distortion
17	malign	malign	small QQINT nodule with irregular contours
18	benign	malign	regular lump
19	benign	benign	absence of pathological findings
20	undefined	benign	underexposed and slightly drawn image

This last work has been realized by a team of expercts from Hospital Sandro Pertini, Rome, that have cooperated together. However due to this collaboration that implies the contribute of many radiologists, it results impossible to quote each name. This last analysis can be considered a confirm that the human prediction works always on the same line. All the analysis confirm the same results, obviously with different comments but the final results is the same. In conclusion it is possible to affirm that for sure ROIs from CBIS-DDSM are difficoult to work on, either for radiologists either for neural networks, if thinking about classification problem. For this reason it appears a good idea to procede with another type of classification that is pixel classification. It will be explained in the next chapter.

Chapter 7

Semantic Segmentation on ROIs

Now some details of the implementation of this part are explained. It has been said many times that *semantic segmentation* is the task of predicting the class of each pixel in an image. An important property is that this problem is more difficult than *object detection*, where the aim is to predict a box around the object. It is slightly easier than **instance segmentation**, since it is necessary not only to predict the class of each pixel but also differentiate between multiple instances of the same class. So the code reported in this section focuses exclusively on *semantic segmentation*. A specific and personal neural network will not be designing, but it will be used *DeepLabv3* with a **Resnet50** backbone from Pytorch âĂŹ s model repository in the way showed in 7.1.

```
from torchvision import models
from torch.utils.data import Dataset, DataLoader
import glob
import os
from PIL import Image
import torch.nn as nn
from torch import optim
import torch
from tqdm import tqdm
from torchvision import transforms
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
import cv2

batch_size = 2
deeplab = models.segmentation.deeplabv3_resnet50(pretrained=0,
                                                 progress=1,
                                                 num_classes=2)
```

Figure 7.1. Model choice from Pytorch âĂŹ s repository

Here model module from torchvision is used to get the *deeplabv3_resnet50* model. In addition the number of classes is specified using *num_classes* as 2 because it will be generated two grayscale images, one for predicting region and another without. The grayscale images will have the same size as the input image. These two images will be compared these predictions to find out whether the model predicts higher chances of hands or no hands at each pixel of the image.
Next, a custom model to process the data is prepared as showed in 7.2:

```
class RoiSegModel(nn.Module):
    def __init__(self):
        super(RoiSegModel,self).__init__()
        self.dl = deeplab

    def forward(self, x):
        y = self.dl(x)['out']
        return y
```

Figure 7.2. ROIs custom model

After that, the dataset is composed. It is given as assumption that masks and images are in separate folders and that those folders are located in the same parent folder. The constructor takes three arguments :

- **parentDir**: The name of the parent folder where image and mask folder are located.
- **imageDir**: The folder where images are located.
- **maskDir**: The folder where segmentation masks are located.

In the constructor for the **SegDataset** 7.3, 7.4 class a list of image and mask filenames is given. In the ___getitem___ function both the image and mask are given. The image is resized and standardized and got the X. For the mask, which is also the label, the image is resized and standardized. After that a ___bitwisenot___ operation is applied on the mask to get another mask that is the exact negative of the original one. Then, those two masks to get a two channel image are stuck together, where the first channel corresponds to the ROIs label and the second one to the no ROIs label.

```
class SegDataset(Dataset):

    def __init__(self, parentDir, imageDir, maskDir):
        self.imageList = glob.glob(parentDir+'/'+imageDir+'/*')
        self.imageList.sort()
        self.maskList = glob.glob(parentDir+'/'+maskDir+'/*')
        self.maskList.sort()
    def __getitem__(self, index):

        preprocess = transforms.Compose([
                                transforms.ToTensor(),
                                ])

        X = Image.open(self.imageList[index]).convert('RGB')
        X = preprocess(X)
```

Figure 7.3. First part of the constructor SegDataset

```
        yimg = Image.open(self.maskList[index]).convert('L')
        y1 = preprocess(yimg)
        y1 = y1.type(torch.BoolTensor)
        y2 = torch.bitwise_not(y1)
        y = torch.cat([y2, y1], dim=0)

        return X, y

    def __len__(self):
        return len(self.imageList)
```

Figure 7.4. Second part of the constructor SegDataset

7.5, 7.6 ar the part of the code related to

```
calcTrain = SegDataset('DatasetInput', 'train', 'ROItrain')
calcTest = SegDataset('DatasetInput', 'test', 'ROItest')

def pad_collate(batch):
    X, y = zip(*batch)
    X = list(X)
    y = list(y)
    maxw, maxh = 0, 0

    for img in X:
        _, w, h = img.shape
        maxw = max(w, maxw)
        maxh = max(h, maxh)
```

Figure 7.5. Dataset part 2

```
    for img in X:
        _, w, h = img.shape
        maxw = max(w, maxw)
        maxh = max(h, maxh)

    for i, (img, roi) in enumerate(zip(X,y)):
        _, w, h = img.shape
        diff_w = maxw - w
        diff_h = maxh - h
        newImg = F.pad(img, (diff_h, 0, diff_w, 0))
        newRoi = F.pad(roi, (diff_h, 0, diff_w, 0))
        X[i] = newImg.unsqueeze(0)
        y[i] = newRoi.unsqueeze(0)
    X =  torch.cat(X)
    y =  torch.cat(y)
    return X, y

train_dataloader = DataLoader(calcTrain, batch_size=batch_size, shuffle=True, collate_fn = pad_collate)
test_dataloader = DataLoader(calcTest, batch_size=batch_size, shuffle=True, collate_fn = pad_collate)
```

Figure 7.6. Dataset part 2

The next step consists in writing some performance metrics to keep track of the training process. In particular two metrics will be used for this task:

- Intersection over Union;
- Pixel Accuracy;

Intersection over Union is a standard metric for segmentation tasks. It gives the area of intersection between prediction and target divided by their union. Intersection over union (IoU) is known to be a good metric for measuring overlap between two bounding boxes or masks. Numerically an IOU value over 0.5 is good.If the prediction is completely correct, IoU = 1. The lower the IoU, the worse the prediction result. In 7.7 this concept is explained.

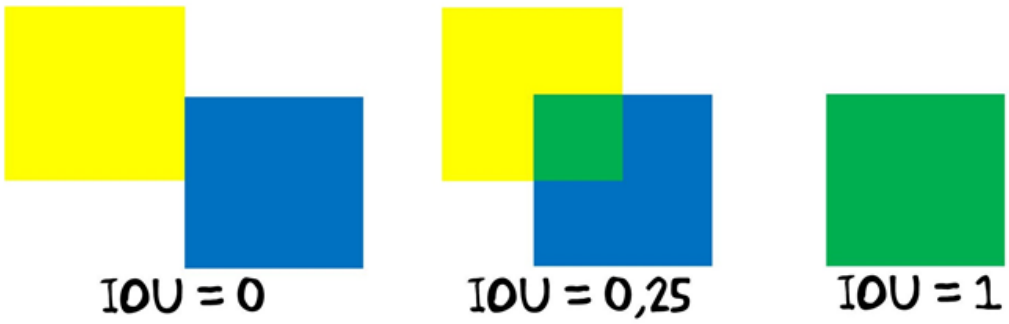

Figure 7.7. Representation of IoU metric

In terms of code the **Intersection over Union** is composed as represented in 7.8:

```
def meanIOU(target, predicted):
    if target.shape != predicted.shape:
        print("target has dimension", target.shape, ",
                predicted values have shape", predicted.shape)
        return

    if target.dim() != 4:
        print("target has dim", target.dim(), ", Must be 4.")
        return

    iousum = 0
    for i in range(target.shape[0]):
        target_arr = target[i, :, :, :].clone().detach().cpu().numpy().argmax(0)
        predicted_arr = predicted[i, :, :, :].clone().detach().cpu().numpy().argmax(0)

        intersection = np.logical_and(target_arr, predicted_arr).sum()
        union = np.logical_or(target_arr, predicted_arr).sum()
        if union == 0:
            iou_score = 0
        else :
            iou_score = intersection / union
        iousum +=iou_score

    miou = iousum/target.shape[0]
    return miou
```

Figure 7.8. IoU implementation

Pixel accuracy is simply the number of correctly predicted pixels divided by total number of pixels. The implementation follows in 7.9

```
def pixelAcc(target, predicted):
    if target.shape != predicted.shape:
        print("target has dimension", target.shape, ",
                predicted values have shape", predicted.shape)
        return

    if target.dim() != 4:
        print("target has dim", target.dim(), ", Must be 4.")
        return

    accsum=0
    for i in range(target.shape[0]):
        target_arr = target[i, :, :, :].clone().detach().cpu().numpy().argmax(0)
        predicted_arr = predicted[i, :, :, :].clone().detach().cpu().numpy().argmax(0)

        same = (target_arr == predicted_arr).sum()
        a, b = target_arr.shape
        total = a*b
        accsum += same/total

    pixelAccuracy = accsum/target.shape[0]
    return pixelAccuracy
```

Figure 7.9. Pixel accuracy implementation

The next phase is the training phase.
The first thing to do consists in creating the optimizer and loss function, model and a learning rate scheduler objects with appropriate hyperparameters. 7.10.

```
model = RoiSegModel()
optimizer = optim.Adam(model.parameters(), lr=0.00005)
loss_fn = nn.BCEWithLogitsLoss ()
lr_scheduler = optim.lr_scheduler.ExponentialLR(optimizer=optimizer, gamma=0.8)
```

Figure 7.10. Creation of the optimizer and loss function, model and a learning rate scheduler objects with appropriate hyperparameters

Later, the training loop function follows (7.11, 7.12, 7.13, 7.14) that will take all these objects created as arguments and train the model. It will display mean loss and the performance metrics during training. After every epoch it will save a checkpoint with all the losses and metrics calculated until now. This will make it possible to stop the training at any time and resume it without any loss of data. The last argument to this function is the path to the checkpoint from which ones wants to resume training. The function returns a tuple of all the important data from the training, loss and metrics on train and validation dataset.

```
def training_loop(n_epochs, optimizer, lr_scheduler, model, loss_fn, train_loader,
        val_loader, lastCkptPath = None):
    if torch.cuda.is_available():
        dev = "cuda:0"
    else:
        dev = "cpu"
    device = torch.device(dev)

    tr_loss_arr = []
    val_loss_arr = []
    meanioutrain = []
    pixelacctrain = []
    meanioutest = []
    pixelacctest = []
    prevEpoch = 0

    if lastCkptPath != None :
        checkpoint = torch.load(lastCkptPath)
        prevEpoch = checkpoint['epoch']model.load_state_dict(checkpoint['state_dict'])
        optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
        for state in optimizer.state.values():
            for k, v in state.items():
                if isinstance(v, torch.Tensor):
                    state[k] = v.to(device)tr_loss_arr =  checkpoint['Training Loss']
        val_loss_arr =  checkpoint['Validation Loss']
        meanioutrain =  checkpoint['MeanIOU train']
        pixelacctrain =  checkpoint['PixelAcc train']
        meanioutest =  checkpoint['MeanIOU test']
        pixelacctest =  checkpoint['PixelAcc test']
        print("loaded model, ", checkpoint['description'], "at epoch", prevEpoch)model.to(device)
```

Figure 7.11. training loop function

```
for epoch in range(0, n_epochs):
train_loss = 0.0
pixelacc = 0
meaniou = 0

pbar = tqdm(train_loader, total = len(train_loader))
for X, y in pbar:
    torch.cuda.empty_cache()
    model.train()
    X = X.to(device).float()
    y = y.to(device).float()
    ypred = model(X)
    loss = loss_fn(ypred, y)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    tr_loss_arr.append(loss.item())
    meanioutrain.append(meanIOU(y, ypred))
    pixelacctrain.append(pixelAcc(y, ypred))
    pbar.set_postfix({'Epoch':epoch+1+prevEpoch,
                      'Training Loss': np.mean(tr_loss_arr),
                      'Mean IOU': np.mean(meanioutrain),
                      'Pixel Acc': np.mean(pixelacctrain)
                     })
```

Figure 7.12. training loop function

```
with torch.no_grad():
    val_loss = 0
    pbar = tqdm(val_loader, total = len(val_loader))
    for X, y in pbar:
        torch.cuda.empty_cache()
        X = X.to(device).float()
        y = y.to(device).float()
        model.eval()
        ypred = model(X)

        val_loss_arr.append(loss_fn(ypred, y).item())
        pixelacctest.append(pixelAcc(y, ypred))
        meanioutest.append(meanIOU(y, ypred))

        pbar.set_postfix({'Epoch':epoch+1+prevEpoch,
                          'Validation Loss': np.mean(val_loss_arr),
                          'Mean IOU': np.mean(meanioutest),
                          'Pixel Acc': np.mean(pixelacctest)
                         })
```

Figure 7.13. training loop function

```
    checkpoint = {
        'epoch':epoch+1+prevEpoch,
        'description':"add your description",
        'state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'Training Loss': tr_loss_arr,
        'Validation Loss':val_loss_arr,
        'MeanIOU train':meanioutrain,
        'PixelAcc train':pixelacctrain,
        'MeanIOU test':meanioutest,
        'PixelAcc test':pixelacctest
    }
    torch.save(checkpoint, 'checkpoints/checkpointhandseg'+str(epoch+1+prevEpoch)+'.pt')
    lr_scheduler.step()

return tr_loss_arr, val_loss_arr, meanioutrain, pixelacctrain, meanioutest, pixelacctest
```

Figure 7.14. training loop function

The last part of the code related to this part, consists in visualize the plots in order to analize the metrics and the results obtained. In fact the aim of the *pixel classification approach* is to find an alternative way to extact significant results with the CBIS-DDSM ROIs images. 7.15 is the part of code of interest for this purpose. First it is possible to see that the model is loaded with the function *torch.load* and the last one of 10 epochs is choosen, this is why checkpointroiseg9.pt is loaded. Checkpoints, as already explained, is a disctionary where all the losses and metrics calculated until now are saved. Then with *plt.subplots* plots are created, in fact this utility wrapper makes it convenient to create common layouts of subplots, including the enclosing figure object, in a single call. The ncols is set as 2 and the nrows is set

as 3 so as a results a 3×2 grid will be shown (7.16). Next, the six plots of interest with the corrispondence labels are instantiated. The six plots are: Training Loss, Validation Loss, MeanIOU train, MeanIOU test, PixelAcc train, PixelAcc test that are the performance metrics characteristics of this kind of problems.

```
retval = training_loop(10,
       optimizer,
       lr_scheduler,
       model,
       loss_fn,
       train_dataloader,
       test_dataloader,
       None)


checkpoint = torch.load('checkpoints/checkpointroiseg9.pt')
fig, ax = plt.subplots(ncols = 2, nrows = 3, figsize = (10,13))
ax[0][0].plot(checkpoint['Training Loss'], 'r', label='training loss')
ax[0][1].plot(checkpoint['Validation Loss'], 'b', label='validation loss')
ax[1][0].plot(checkpoint['MeanIOU train'], 'g', label='meanIOU training')
ax[1][1].plot(checkpoint['MeanIOU test'], 'r', label='meanIOU validation')
ax[2][0].plot(checkpoint['PixelAcc train'], 'b', label='pixelAcc  training')
ax[2][1].plot(checkpoint['PixelAcc test'], 'b', label='pixelAcc validation')
for i in ax:
    for j in i:
        j.legend()
        j.grid(True)
plt.show()
```

Figure 7.15. Extraction of plots

In 7.16 plots are reported. The first characteristics that these plots highlight are that the loss, both for training and validation, decrease a lot in particular in the training loss plot, the first one. In the second one, the validation loss plot there is an important decrease, with the difference of a small peak in the final part.
The second important characteristic is found in the second couple of plots, the Intersection over Union (MeanIOU) one. In the case of the training the plot always increases, it seems to be line. In the case of the validation the whole plot increases without any doubts but not in a direct way as the training one. The last couple of plots represents the Pixel Accuracy. As the two previous case the plot related to the training increases, on the other hand the plot related to validation increase but in an unstable way.
The results analyzed, in particular the analysis on the validation set, hightlights that there is an actual aknowedge of the information, so the problem encountered with the image classification is solved and consistent results have been extracted from the CBIS-DDSM Rois dataset.

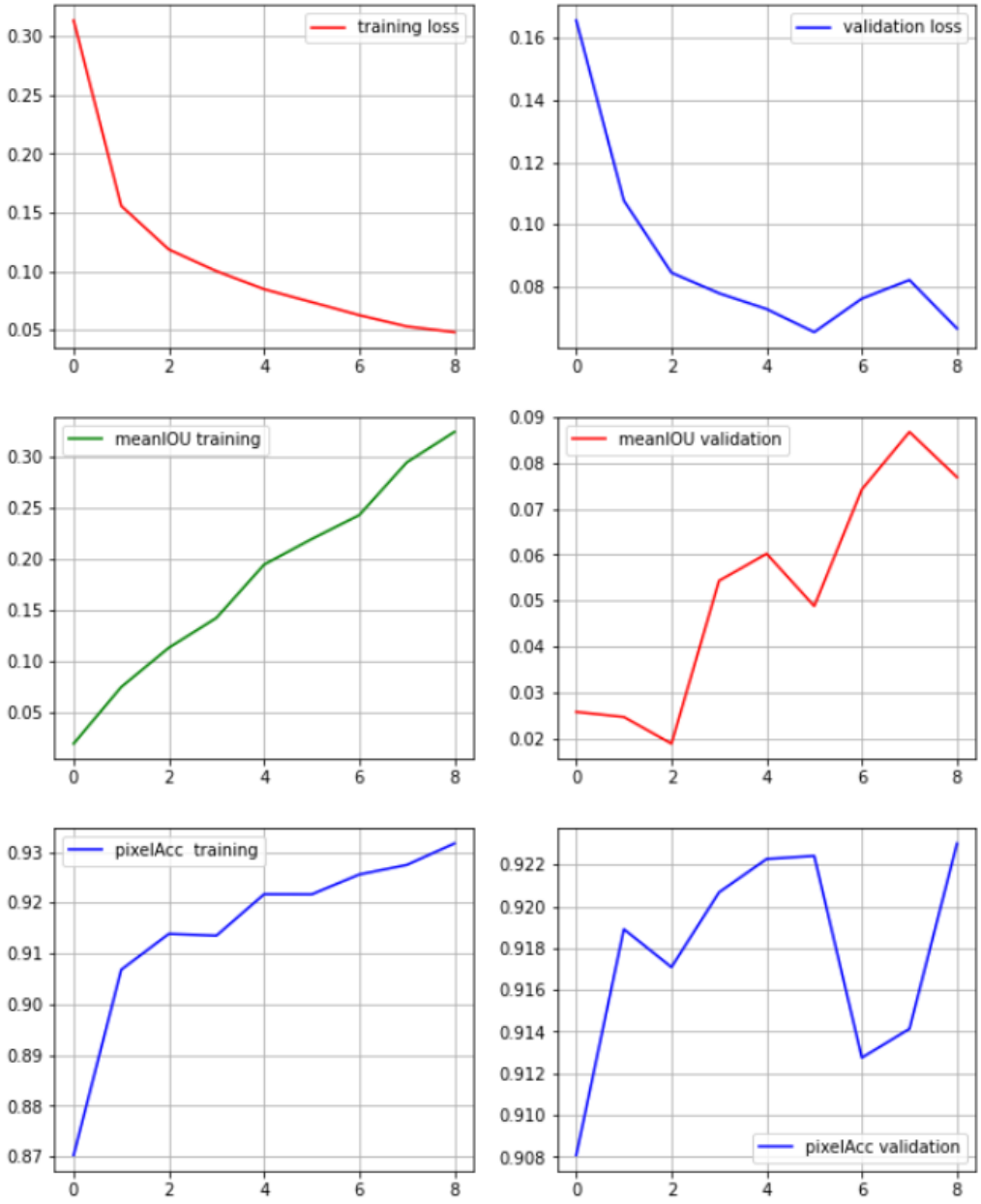

Figure 7.16. Aims of image segmentation for medical images

Chapter 8

Conclusion

The aim of this project consisted in apply Deep Learning and all its techniques in the field of breast cancer. The importance of Deep Learning in this delicate field is related to the fact that there may are subtle but very informative cues on mammograms that may not be discernible by humans and deep learning can leverage these cues to yield improved risk models. So Deep Learning represent an objective tool to face up this problematic. In fact until now, the ways to prevent breast cancer consist in the visual inspection of mammograms images that should enable radiologists to examine and identify tissues carefully, looking for malignant lesions. Malignant lesions are mainly in the form of clusters of microcalcifications and masses. However, unfortunately, the evaluation of mammograms is a complicated and time consuming process so Deep Learning represent an innovative way to prevent the problem. Of course other tools are available nowadays, such as Computer-aided detection and computer-aided diagnosis that are known as CAD systems; they are used by radiologist in the process of diagnosis. They interpret digitally captured medical images and provide useful information about the suspicious regions with the help of artificial intelligence and image processing techniques for the task of detection and analysis.
After having in mind the information stated above, the first track that seemed to be promising was *image classification* on CBIS-DDSM so a division beetwen malignant and benignant tumors. In a second moment this analysis was improved considering ROIs, provided by CBIS-DDSM. ROIs are litterally region of interest and they represent the part of the whole mammogram, that contains the lesion from which it is possible to state if an image belongs to a benignant or a malignant tumors. However, this analysis presented some problematics that needed to be faced up. In fact it seemed that all the models choosen for this purpose did not learn at all or they predicted 0 or 1 withouth any criteria. Two solution were thought in the hope of leading to a final solution: the use of Explainable AI, the analysis of 5 radiologists of a set of 20 images from CBIS-DDSM ROIs. At the end what arised was that the dataset at hand was not appropriate for this kind of problem, image classification. The solution was to change the type of problem, going from an image classification problem to a pixel classification analysis. The output of this problem was of course important information for the analysis of breast cancer but instead of recognize beetwen benignant and malignant tumors, the idea was, once the ROIs is given, to

recognize the pixels related to the lesions. This new kind of problem is very useful if one thinks that a radiologist's analysis can be improved since the time consuming is reduced. Of course another solution could have been to search for another dataset but CBIS-DDSM presented many characteristics that made it appropiate for such a work. In fact the amount of data and the fact that the dataset is balanced made it the best dataset to work on with.

Appendix A

Technical details about scripts

A.1 Conversion_dataset.py

```
import numpy as np
import pydicom
from PIL import Image
import os

def get_names(path): #path lavora in directory corrente
    names = []
    for root, dirnames, filenames in os.walk(path):
        for filename in filenames:
            _, ext = os.path.splitext(filename)
            if ext in ['.dcm']:
                names.append(filename)
    return names

def convert_dcm_png(name): #converte in png

    im = pydicom.dcmread(name)

    im = im.pixel_array.astype(float)

    rescaled_image = (np.maximum(im, 0)/im.max())*255 #pixels' float
    final_image = np.uint8(rescaled_image) #int pixels
    final_image = Image.fromarray(final_image)
    return final_image

names = get_names('MassTest')
for name in names:
    image = convert_dcm_png('MassTest/'+name)
    name = name.rsplit(".", 2)[0]
    image.save("PNGOutput/"+ name + '.png')
```

The code of **Conversion_dataset.py** consists in two main functions, and as already explained it is used to convert the dataset composed of DICOM files into a png one. The two functions are:

- def get_names(path);
- def convert_dcm_png(name);

It is important to highlight some important aspects.
The code has been thought as to work **in the current directory**. It means it is not possible to change the path manually and it is necessary to put **Conversion_dataset.py** in the same directory of the dataset in DICOM.
Then an empty list named *names* has been created and after it a loop passing through all the directories, or images or names of the images. In fact the data are three:

- root;
- dirnames;
- filenames;

In this particoular case *filenames* is what matters but **os.walk()** generates the file names in a directory tree by walking the tree either top-down or bottom-up. For each directory in the tree rooted at directory top (including top itself), it yields a 3-tuple (dirpath, dirnames, filenames).

- root: prints out directories only from what specified;
- dirnames: prints out sub-directories from root;
- filenames: prints out all files from root and directories;

This is why both root and dirnames appear in the code too.
Next in the code there is another loop that will give as result **only** DICOM images. The previous one does not distinguish between DICOM type and other extension type so another loop is necessary.
Then, **os.path.splitext** *splits* the name of each file into two part, one is the name, one is the extension. Since the only type needed is **.dcm**, a simple *if* follows that check if the extension actually corresonds to .dcm and if so add it to the empty list *names* created at the begining of the function.

The second function of this script, **convert_dcm_png**, as its name suggests, aims to convert a DICOM image into a png image.
First, from **pydicom** which is a library from Python useful to work with dicom datasets, the function **pydicom.dcmread** just read and parse a DICOM dataset stored in the DICOM file format.
Then **pixel_array.astype** converts to float to avoid overflow or underflow losses.
The next lines just work on pixels and convert the image into a png one. This process is necessary because dicom images are 16 bit images that can t be opened by image viewer, we want to transform them into 8-bit images.

The final part of this script performs just few functions. First, it is possible to see that it is a simple loop on the dataset. Then the function **convert_dcm_png** is called to compute the conversion.
Before saving the results it has been useful to insert a **rsplit** in order to obtain the images in png in this form: file1.png. Without the **rsplit** the result would

have been file1.dcm.png. All the png images have been stored in a new file called **PNGOutput**.

A.2 MassTrainingGenpkl.py

```
import csv
import collections
import pickle
path ="/media/federica/Loserino/CBIS-DDSM/manifest-Egq0PU078220738724328010106/Mass-Training/"
ifile  = open(path+"metadata.csv", "r")
read = csv.reader(ifile)
numeri = []
for row in read :
    if row[4] == "Subject ID":
        continue
    #print (row[4])


    l = row[4].split("_")
    #valore_fisso = l[1];
    numero = l[2];
    numeri.append(numero)
numeri.sort()



res = [item for item, count in collections.Counter(numeri).items() if count == 4] #list comprehension genera lista a partire da un' altra lista
#print(len(res))
esami = []
#[{'horizontal_flip': 'NO', 'L-CC': ['0_L_CC'], 'L-MLO': ['0_L_MLO'], 'R-MLO': ['0_R_MLO'], 'R-CC': ['0_R_CC']}
for n in res:
    esame = {'horizontal_flip': 'NO',
                    'L-CC': [n+"_LEFT_CC"],
                    'L-MLO': [n+'_LEFT_MLO'],
                    'R-MLO': [n+'_RIGHT_MLO'],
                    'R-CC': [n+'_RIGHT_CC']
                    }
    #print(dizionario)
    esami.append(esame)
print(esami)


file_name = path + 'exam_list_before_cropping.pkl'
with open(file_name, 'wb') as handle:
    pickle.dump(esami, handle) #crea file binario con dentro scritto esami
```

MassTrainingGenpkl.py is a very **important** file. In fact it generates a binary file called **exam_list_before_cropping.pkl** that is fondamental for the **cropping phase** since **crop_mammogram** takes as input this file.pkl. The first part of this script focuses on reading a csv file, *metadata.csv*.

The first thing to notice is the path of the directory that is defined manually. This is the opposite of what happens in **Conversion_dataset.py** where the path is not defined and the file needs to be inserted in the same directory of the dataset. Subsequentially the csv is open and read with **csv.reader**.
The fifth col of the csv is the most important since the numbers of the exams are stored there. So the code focuses on reading just the values in this col and this is the meaning of the following for, **for row in read**. Next, the if control aims just to skip the first element of the col. In this specific case it corresponds to *Subject ID* and it is not relevant for the reserch.
The **split** divides the string in parts according to the symbol _ in the following way. The strings in the fifth col is in the form:
Mass-Training_P_00001_LEFT_CC
Dividing the string by _ 5 substrings are obtained:

- Mass-Training;
- P;
- 00001;
- LEFT;
- CC;

Remembering that the count starts from 0 indices it is easy to see that: Mass-Training corresponds to l[0];
P corresponds to l[1];
00001 corresponds to l[2];
LEFT corresponds to l[3];
CC corresponds to l[4];
As a consequence the part of the string in the third position is stored as the variable **numero**.
Now it is good to analize a part of the code that has been deleted from the final version but has been fondamental to lead to a brillant solution of the problem.
In fact in the first version of this script before reading the fifth col an empty list was created and filled with **numero**, and the again ordered with a **sort**. Next, another important thing to say is that in the first moment the idea was to consider **only** the exams with all the four views. This has been implemented with a **list comprehension**. A **list comprehension** generates a list from another list.
Then there is the creation of a dictionary with five keys:

- horizontal_flip;
- L-CC;
- L-MLO;
- R-MLO;
- R-CC;

This structure highlights the fact that an exam, in order to be considered, must have all the four views. For this reason the total number of exams that actually corresponds to this requirement was very low. Only 88 images on around 6000, considering also the test and not only the training, resulted to be analizable. Clearly it was a problem. The dataset resulted quite inconsistently. This result has been stored in the empty list *esami*, filled with an *append*.
Due to the small dimension of the dataset, **MassTrainingGenpkl.py** has been modified.
Since a new script has been created, this file presents the same first part where the csv is read. Also the *split* phase is the same.However the differences are quite a lot. First **esami** is an empty dictionary, whereas in **MassTrainingGenpkl.py** it was a list.
Then after the split there is an *if* that check if *numero* is not yet in the dictionary *esami*. This is a very important aspect, in fact if the current number is not present in the dictionary, it is necessary to set the new *key* corrisponding to *horizontal_flip*. If the number is still present this passage is not necessary so this is the meaning of that if.
Subsequently for both cases two settings are needed: **nome_esame** is obtained considering the split done at the begining of the code, but now taking l[3]. l[3] in the csv can assume two different values:

- LEFT;
- RIGHT;

just the first character is important so this is why l[3][0] and not just l[3]. On the other hand l[4] in the csv can assume two different values:

- CC;
- LMO;

so the result of *numero_esame = l[3][0] + '-' + l[4]* can be:

- L_CC;
- L_LMO;
- R_CC;
- R_LMO;

Finally, the dictionary is filled with the last key that returns a value composed of the number of the exam and the specific view.
Last, there is a convertion in list of all the *values* of the dictionary.
The last part of this script focuses on creating the pkl file.

8.3 convert.sh

```
#!/bin/bash

find . -mindepth 2 -type f | while read file; do
    # echo "$file"
    newname=${file:2}
    newname=${newname//\//_}
    cp "$file" ./"$newname"
done
```

The script **convert.sh** is a very short but important script in *bash*.
When the dataset has been download all its file were stored with very long and unclear names. In order to build scripts able to simplify the analysis of the dataset, it has been necessary to make the whole structure simpler.
The first problem that this script has solved was related to the fact that the original structure of the dataset provided more than two folders for each image, in form of subfolders. This specific structure was problematic since the user was forced to change manually one file at the time. Clearly this is not a problem for very small datasets but it is for the one at hand.
The first line of the script faces this issue. In fact *mindepth 2* tells to process only files that are at least depth 2.
| in bash passes the result of an operation, in this case the *mindepth*, to another one, in this case the *loop*.
The next line considers the name of the file as a string without the first two characters that are ./
The next operation changes the character /in _ in order to *delete* the folders.
Last, the comand *mv* substitutes **file** into **filename**.
It is possible to see the original structure of the dataset before the execution of **convert.sh** in the following image

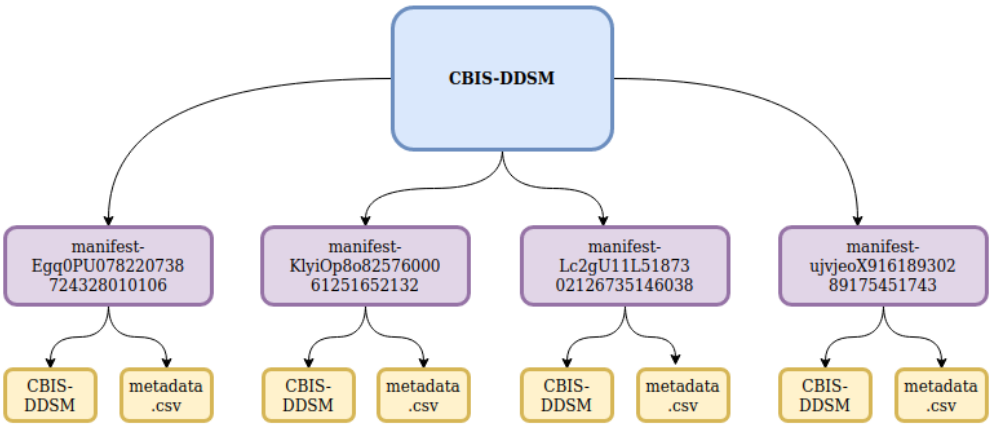

Figure 8.1. first part of the structure of CBIS-DDSM

CBIS-DDSM highlighted in blue in 8.1 is the main folder. As it is possibile to see in the scheme, another folder called *CBIS-DDSM* exists in every of the four subfolders. For semplicity the structure of these second folders called *CBIS-DDSM* is represented in another picture few lines later.

Now in 8.1 in the main folder there are four folders:

- manifest-Egq0PU0782207387243280101 06;
- manifest-KlyiOp8o82576000612516521 32;
- manifest-Lc2gU11L5187302126735146038;
- manifest-ujvjeoX91618930289175451743;

These names are the names when the dataset is downloaded but every foulders contains four different type of images:

- Mass Training;
- Mass Test;
- Calc Training;
- Calc Test;

In each of these four folders there are the following files:

- CBIS-DDSM;
- metadata.csv

The content of CBIS-DDSM is explained now in some details, **metadata.csv** is a csv and contains the information of all the files stored in CBIS-DDSM.
CBIS-DDSM has the structure explained in 8.2:

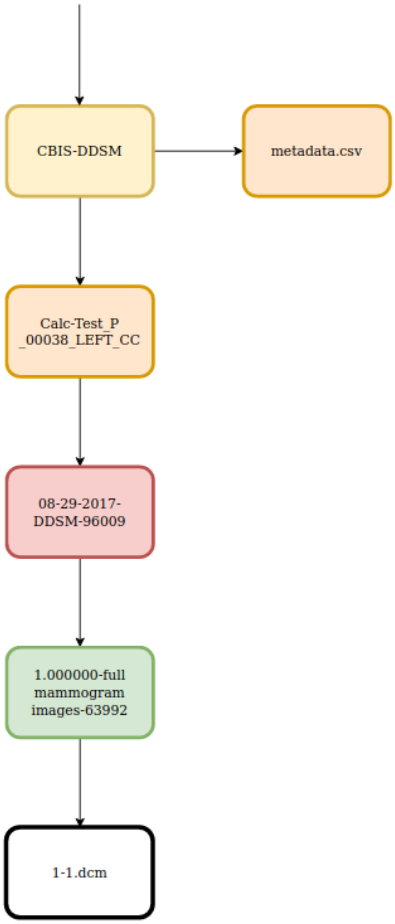

Figure 8.2. this structure shows the depth of every dcm file before applying any scripts

Starting from 8.1, in 8.2 illustrates that in the folder *CBIS-DDSM* there are many other folders before arriving at the actual dcm file. It is easy to see that the original composition of the dataset is very articulated. For this reason the script **convert.sh** has been very useful. After applying **convert.sh** the new structure consists consists in two folders and a python file in each of the four initial folder:

- MassTest;
- PNGOutput;
- conversion_dataset.py;

MassTest contains the images with *dcm* extension.
PNGOutput contains the same images as **MassTest** but with *png* extension, this is the output of **conversion_dataset.py**.
Last, there is **conversion_dataset.py** that as said, must be placed in the same directory of the dataset.
An important thing that 4.7 highlights is that both the names of files into *MassTest* and *PNGOutput* are very simplified. This semplification is actuated through another script in bash, **simplify.sh**.

8.4 simplify.sh

```
#!/bin/bash

for file in ./*.dcm; do #ciclo su tutti i file.dcm
  newfile=‘echo "$file" | cut -d "_" -f 3-5‘ #rinomino file in
    newfile passo il risultato a cut che fa come una split e conto da
    3 a 5 in base a _
  #echo "$newfile"
        mv "$file" $newfile.dcm #rinomino

done
```

The first line is a loop on all the files.dcm.
Then *file* is renominated as *newfile*, | passes the result of this operation to the next one, the *cat* that acts like a *split* according to the character _. The part of the string of interest goes from 3 to 5.
Last **newfile.dcm** is renominated through a *mv* into file. This operation is very important, because it deletes the extension .dcm. Otherwise the result would be *00016_LEFT_CC.dcm.png*.
At this point everything is almost complete to compute the **cropping**. However a small modification in **data_handling.py** occurs. In fact this file is related to the generation of the *pkl* file. It turns *exam_list* from **MassTrainingGenpkl.py** into *image_list* for parallel functions which process each image separately.
The code is composed of two functions:

- unpack_exam_into_images;
- add_metadata;

The problem to solve was related to the fact that not every exam contains the four view. For this reason an if control has been added to the two functions. The control added consisted in checking if the index *view* was not present among the keys of the dictionary.
The same control has been inserted in the second function.

8.5 labelstraining.py

```
import csv
import collections
```

```
import pickle
import os
import shutil
path ="/home/federica/Desktop/TesiMagistrale/sample_output_mass_test/
    "
ifile  = open(path+"mass_description_test.csv", "r")
read = csv.reader(ifile)



for row in read :
  if row[9] == "pathology":
    continue
  if row[9] == "BENIGN":
    newpath = r'/home/federica/Desktop/TesiMagistrale/
    sample_output_mass_test/benign'
    if not os.path.exists(newpath):
      os.makedirs(newpath)
  if row[9] == "MALIGNANT":
    newpath = r'/home/federica/Desktop/TesiMagistrale/
    sample_output_mass_test/malign'
    if not os.path.exists(newpath):
      os.makedirs(newpath)
  if row[9] == "BENIGN_WITHOUT_CALLBACK":
    newpath = r'/home/federica/Desktop/TesiMagistrale/
    sample_output_mass_test/benign_without_callback'
    if not os.path.exists(newpath):
      os.makedirs(newpath)
  original = "/home/federica/Desktop/TesiMagistrale/
    sample_output_mass_test/cropped_images/" +row[0][2:] + "_" + row
    [2] + "_" + row[3] + ".png"
  target = newpath
  try:
    shutil.move(original,target)
  except:
    print('esiste giÃ!')
```

labelstraining.py is a very important script for the training phase. The entire script is based on the csv files related to both *training* and *test* for *Mass* and *Calc*. So there will be four different csv.
For a first attempt only **Mass** has been considered, and the script in **??** refers to test, but it is exactly the same for training, it is just necessary to change the variable *path* at the beginning.
Next the csv is read and the column 9 is considered since it contains the labels of the dataset. As usual the first if just skip the first line that is equal to a target of the column, in this case **pathology**.
The labels of the dataset are:

- benign;
- malign;
- benign_without_calling;

At first all the three labels have been differentiated. For this reason in the code there are three if controls that creates a new folder for each labels if this has not

been created before, through the *os* library.
Finally with another library, *shutil* each image, according to its label is moved to the right folder.
Now everything is ready to compute the training. The code was thought in a first moment for a **local training** but then out of some memory problems it was changed and moved on colab. After many imports there is an important function added later because of a problem. All the images in the dataset had different dimensions and this fact generated an error during the training. The function **pad_collate** faces this problem since the output will be a set of all images with the same dimensions. The function takes in input *batch* that is a list of tuples where the first element is the **image** and the second is the **class**. With *zip(*batch)* the tupla is transformed into a list, so there are two lists:

- one for the images, a list of *tensor*;
- one for the classes;

Next in the code the core of the function is implemented. In fact the aim of **pad_collate** is to transform the whole dataset in the same dimensions and this is achived transforming the smaller images in the same dimensions as the biggest one. The reason why the biggest images have not been transformed basing on the smallest image is related to the fact that redusing the dimension of an image the risk to lose important parts is high. On the other hand making bigger a small image does not result in loss of information.
After the *zip* the list of images has been saved as X and will be very useful for loop on all images later. Moreover two new variables are created, *maxw* and *maxh* that rispectly indicates the maximum value of the weight and the maximum value of the height.
img.shape gives as result the component of each tensor:

- number of channels;
- weight;
- height;

The first parameter is skipped since it is not relevant. Then the first loop on X set values to *maxw* and *maxh*. Then the differences between *maxw* and *w* is computed and also the differences between *maxh* and *h*. *F.pad* adds padding to each image and optionally on each dimension of the image. This is done in order to obtain all the same dimensions. In the specific case the padding is of 0 padding that perfectly match with the dataset since the images have already black background.
Subsequentially *torch.unsqueeze* is needed since it returns a new tensor with a dimension of size one inserted at the specified position. This solved the problem related to the uncompatible dimensions of tensor, the input had 3 dimensions as said before but 4 was needed. *torch.cat* concatenates the given sequence of tensors in the given dimension, remembering that all tensors must either have the same shape or be empty.

Finally *torch.tensor* constructs a tensor with data in y in the current case it is equal to three. Next just a training in pytorch follows.

Bibliography

M. Al-Qizwini, I. Barjasteh, H. Al-Qassab, and H. Radha. Deep learning algorithm for autonomous driving using googlenet. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 89–96. IEEE, 2017.

S. A. Alanazi, M. Kamruzzaman, M. N. Islam Sarker, M. Alruwaili, Y. Alhwaiti, N. Alshammari, and M. H. Siddiqi. Boosting breast cancer detection using convolutional neural network. *Journal of Healthcare Engineering*, 2021, 2021.

N. F. Boyd, L. J. Martin, M. Bronskill, M. J. Yaffe, N. Duric, and S. Minkin. Breast tissue composition and susceptibility to breast cancer. *Journal of the National Cancer Institute*, 102(16):1224–1237, 2010.

F. Bray, J. Ferlay, I. Soerjomataram, R. L. Siegel, L. A. Torre, and A. Jemal. Global cancer statistics 2018: Globocan estimates of incidence and mortality worldwide for 36 cancers in 185 countries. *CA: a cancer journal for clinicians*, 68(6):394–424, 2018.

H.-P. Chan, S.-C. B. Lo, B. Sahiner, K. L. Lam, and M. A. Helvie. Computer-aided detection of mammographic microcalcifications: Pattern recognition with an artificial neural network. *Medical physics*, 22(10):1555–1567, 1995.

T. G. Debelee, A. Gebreselasie, F. Schwenker, M. Amirian, and D. Yohannes. Classification of mammograms using texture and cnn based extracted features. *Journal of Biomimetics, Biomaterials and Biomedical Engineering*, 42:79–97, 8 2019. doi: 10.4028/www.scientific.net/JBBBE.42.79.

G. Gonella, M. Paracchini, E. Binaghi, and M. Marcon. Breast lesion detection from mammograms using deep convolutional neural networks. In *Proceedings of the 2020 European Symposium on Software Engineering*, pages 120–124, 2020.

R. M. Haralick and L. G. Shapiro. Image segmentation techniques. *Computer vision, graphics, and image processing*, 29(1):100–132, 1985.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks, 2018.

J.-J. Hwang and T.-L. Liu. Pixel-wise deep learning for contour detection. *arXiv preprint arXiv:1504.01989*, 2015.

A. Krizhevsky. One weird trick for parallelizing convolutional neural networks, 2014.

E. Le, Y. Wang, Y. Huang, S. Hickman, and F. Gilbert. Artificial intelligence in breast imaging. *Clinical radiology*, 74(5):357–366, 2019.

X. Liu, L. Song, S. Liu, and Y. Zhang. A review of deep-learning-based medical image segmentation methods. *Sustainability*, 13(3):1224, 2021.

N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018.

C. Matsoukas, A. B. I. Hernandez, Y. Liu, K. Dembrower, G. Miranda, E. Konuk, J. F. Haslum, A. Zouzos, P. Lindholm, F. Strand, and K. Smith. Adding seemingly uninformative labels helps in low data regimes, 2020.

A. McGuire, M. Drummond, M. Martin, and N. Justo. End of life or end of the road? are rising cancer costs sustainable? is it time to consider alternative incentive and funding schemes? *Expert review of pharmacoeconomics & outcomes research*, 15 (4):599–605, 2015.

A. A. Mohamed, W. A. Berg, H. Peng, Y. Luo, R. C. Jankowitz, and S. Wu. A deep learning method for classifying mammographic breast density categories. *Medical physics*, 45(1):314–321, 2018.

N. R. Pal and S. K. Pal. A review on image segmentation techniques. *Pattern recognition*, 26(9):1277–1294, 1993.

D. L. Rubin. Cbis-ddsm: Acurated. 2018.

A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

K.-H. Shih, C.-T. Chiu, J.-A. Lin, and Y.-Y. Bu. Real-time object detection with reduced region proposal network via multi-feature concatenation. *IEEE transactions on neural networks and learning systems*, 31(6):2164–2173, 2019.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision, 2015.

S. A. Taghanaki, K. Abhishek, J. P. Cohen, J. Cohen-Adad, and G. Hamarneh. Deep semantic segmentation of natural and medical images: A review, 2020.

C. Urbaniak, G. B. Gloor, M. Brackstone, L. Scott, M. Tangney, and G. Reid. The microbiota of breast tissue and its association with breast cancer. *Applied and environmental microbiology*, 82(16):5039–5048, 2016.

A. Yala, C. Lehman, T. Schuster, T. Portnoi, and R. Barzilay. A deep learning mammography-based model for improved breast cancer risk prediction. *Radiology*, 292(1):60–66, 2019.