Contents

1 Introduction

This project includes various analyses and classifications related to mammogram images.
You can refer to these *ipynb* files to verify the code we worked on:

Task 2.1:

- *Scratch_ CNN_ 2.1.1 (no data aug)* is the first implementation
- *Scratch_ CNN_ 2.1.2 (data aug)* is the implementation with data augmentation
- *Scratch_ CNN_ 2.1.3 (holdout val)* is the final validation of the best model and the training used for saving the model "scratch_cnn_1"

Task 2.2:

- *Scratch_ CNN_ 2.2.1 (data aug)* is the implementation with data augmentation
- *Scratch_ CNN_ 2.2.2 (holdout val)* is the final validation of the best model and the training used for saving the model "scratch_cnn_2"

Task 3.1:

-

Task 3.2:

-

Task 4:

- *Baseline_ CNN_ 1* is the implementation of a classification between mass and calcification, using also the baseline patches

Task 5:

- *Ensemble_ 1* is the implementation of the ensemble learning for the first classification
- *Ensemble_ 2* is the implementation of the ensemble learning for the second classification

2 The dataset

The data on which we work are based on the **CBIS-DDSM** (Curated Breast Imaging Subset of Digital Database for Screening Mammography): it is a set of scanned film mammography studies, adapted to carry out an **abnormality diagnosis classification**. Following the detailed description attached to each original image, they can be divided into four classes that distinguish whether the represented abnormality patch is a mass or a calcification, and again whether it is benign or malignant.
From each original image, the abnormality patch was extracted, and in addition a patch of healthy tissue, adjacent to each abnormality patch, was also extracted. Both abnormality patches and baseline patches have been resized to shape $(150x150)$ and have been added to the final images tensor, that counts 5352 elements (2676 abnormality patches and 2676 baseline patches). Finally class labels have been assigned to the patches according to the following mapping:

- 0: Baseline patch
- 1: Mass, benign
- 2: Mass, malignant
- 3: Calcification, benign
- 4: Calcification, malignant

3 Task 1: Related works

1 Transfer Learning and Fine Tuning in Breast Mammogram Abnormalities Classification on CBIS-DDSM Database (Lenin G.Falconi, Maria Perez, Wilbert G.Aguilar, Aura Conci) 2020

2 Deep Learning for Breast Cancer Diagnosis from Mammograms - A Comparative Study (Lazaros Tsochatzidis, Lena Costaridou, Ioannis Pratikakis) 2019

3 Automatic Mass Detection in Mammograms Using Deep Convolutional Neural Networks (Richa Agarwal, Oliver Diaz, Xavier Lladó, Moi Hoon Yap, Robert Martí) 2019

4 Multi-View Feature Fusion Based Four Views Model for Mammogram Classification Using Convolutional Neural Network (Hasan Nasir Khan, Ahmad Raza Shahid, Basit Raza, Amir Hanif Dar, Hani Alquhayz) 2019

The first paper, working on our same dataset, deals with the classification of abnormalities using ImageNet pretrained convolutional networks, with the technique of transfer learning (TL) but also fine tuning (FT). The authors specify that the main obstacle for this kind of problem is that there are not yet mammogram public datasets that are quite extensive and "deep", and consequently if we have few data it is difficult to avoid overfitting. The proposed approach, both with TL and FT, consists in replacing the last fully connecting layer related to the original ImageNet classification task with a set of layer: Global Average Pooling 2D, Full Connecting, Dropout, and Classification Layer. They focused on ìly on the masses images, that are initially pre-processed to enhance contrast, then normalized between 0 and 255, and finally rearranged by data augmentation. The dataset was increased to a total of 60.000 images, where 80% are used for training, 10% for validation, and 10% for testing. Looking at the results of TL, they obtained the best AUC with the Resnet-50, that although overfitted. On the other hand, both VGG16 and VGG19 did not presented overfitting but their classification performance were lower. Regarding to FT instead, the best resultes were obtained by VGG16, unfreezed once from the 8th layer and once from the 10th layer, and by VGG19, unfreezed from the 17th layer.

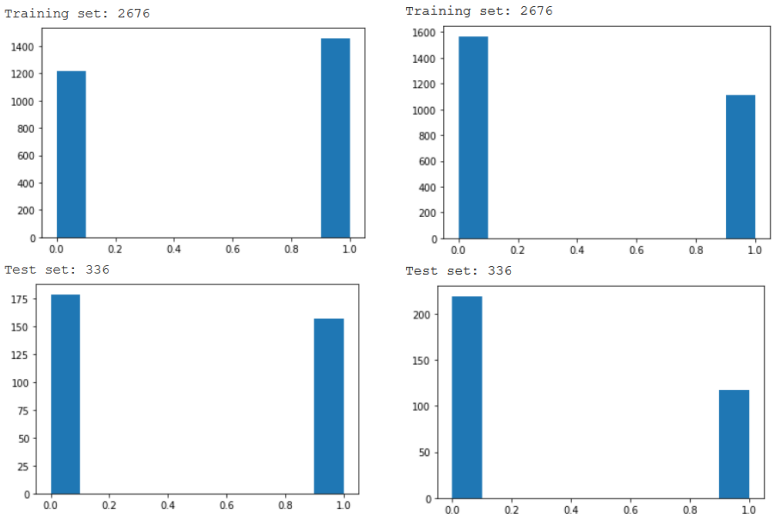
In the second paper, the authors worked on both the CBIS-DDSM dataset and another dataset, the DDSM-400, again to perform a classification of masses between benign and malignant. The initial approach presented is based on training networks from scratch (starting from the Glorot initialization), while later an approach using networks pretrained on the ImageNet datased is presented. They used data augmentation, in particular rotation and flipping transformations, performed online. The best results were achieved with CNNs trained under the fine tuning scenario: maximum performance is found with Resnet-50 and Resnet-101. Regarding the networks from scratch, the authors explain that the lowest results are due to the limited number of samples in the analyzed dataset: they say that in these cases “effective training of larger and complex networks cannot be achieved”.

In the third paper, a patch detector was initially developed, again from the CBIS-DDSM dataset, giving as input to the tested networks patches of abnormal and baseline tissue extracted from mammograms. The authors have used the pretrained networks VGG16, Resnet-50, and InceptionV3, with a transfer learning approach. Since the available images were in black and white, they were replicated on three channels to be compatible with the ImageNet pretrained networks. In addition, the data were augmented on-the-fly using horizontal flipping, rotation, and rescaling. The results of training the CNN on CBIS-DDSM demonstrated that the feature domain of the CNN can be well adapted from natural images to classify masses in mammograms. The best performance were obtained with the InceptionV3.

The last paper were studied mostly for the preprocessing of the data. The experiments in this paper are carried on the CBIS-DDMS and mini-MIAS public databases. First of all, the authors performed some pre-processing steps like rescaling, bilateral filtering, contrast enhancement, on the data for achieving better performance. Then they applied data augmentation transformations to minimize the effect of overfitting, getting a dataset seven times bigger than the original one. Starting from some CNN architectures for training and fine tuning, the authors made adaptations for classification of mammograms on different levels (i.e. normal/abnormal, mass/calcification, and malignant/benign).

4 Data Preprocessing

In this section we describe the main preprocessing applied to the data before the training of the models. Starting from the numpy arrays of the dataset images we were given, first of all we deleted all the samples associated with the *baseline patch* label. Since this labeled images were placed in the even positions of the dataset, we just selected all the odd-index samples and discard the others. This is done both for the training data and the test data.
Then we aggregated the labels according to the classification that we needed to do: in the *Task* 2.1 and 3.1 the classes *mass benign* and *mass malignant* were aggregated in a unique class *mass*, and so for the *calcification* classes. On the other hand, in the *Task* 2.2 and 3.2 we aggregated *mass benign* and *cacification benign* in the *benign* class, and the other labels in the *malignant* class.



(a) *Task* 2.1: 0 for *mass*, 1 for *calcification*

(b) *Task* 2.2: 0 for *benign*, 1 for *malignant*

Figure 1: Label distribution

For the first case, we can see from the figure that the labels are pretty much equally distributed. Instead, in the second case the dataset is a little more unbalanced towards the *benign* class.
Finally, we reshaped all the training images tensors in a single *numpy* array, that has been then normalized. The same has been done for the test images.

5 Task 2: From Scratch CNN

We developed different models of neural networks to work with the given dataset from scratch and we did different tests to find the better hyperparameters to build the final model. This section is divided in two parts: i ìn the first one we show how we built a model for classifying the dataset images between *mass* abnormality and *calfication* abnormality, while in the second part we describe the model built to classify between *benign* and *malignant* abnormalities.

5.1 Task 2.1: Mass-Calcification discriminator

We first tried to build a simple CNN, with some convolutional and max-pooling layers, and two final dense layers. We trained and tested the model several times, changing the hyperparameters to find a good model for our problem. The final architecture that has been used for the most relevant tests is shown in figure 2. It is made of five convolutional layers, with an increasing number of kernels. Each layer is followed by a max-pooling of $2x2$, except for the last one so that the input of the dense layer is not too reduced. The activation function used in the layers is *ReLu*.

After asserting the architecture, the model has been trained with different numbers of *hidden units* for the first fully connected layer, but also for different *batch sizes*, different *output functions* and different *optimizers*. In all the tests, the conclusion has been that the network was over-training, and this is the reason for which we have developed a second model, adding a data augmentation

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 128)         73856
_________________________________________________________________
flatten (Flatten)            (None, 3200)              0
_________________________________________________________________
dense (Dense)                (None, 64)                204864
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 343,842
Trainable params: 343,842
Non-trainable params: 0
```

Figure 2: Architecture of the CNN from scratch (*Task* 2.1)

preprocessing. In the figure 29, we present the results in term of accuracy and loss, but also the consufion matrix (fig. 30), related to a test of the model with 64 hidden units, a batch size of 25, and the *softmax* function for the output layer. The used optimizer was *Adam*, with the deafult learning rate of 0.001. As we said before, the gap between the training and validation graphs shows that the model is over-training: even if the predictions on the test set are accurate, we cannot rely on this network as it is.

	Training set	Validation set	Test set
accuracy	0.9982	0.8134	0.8244
loss	0.0123	1.1618	1.5748

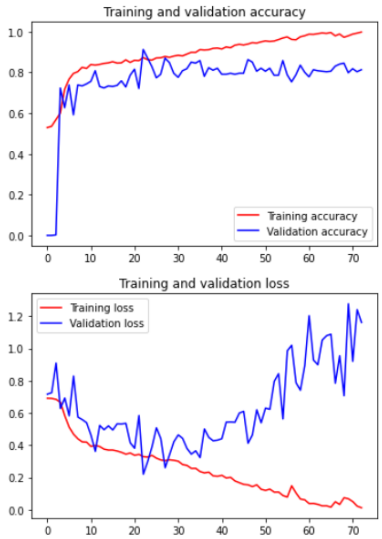


Figure 3: Accuracy and loss graphs for the first model training

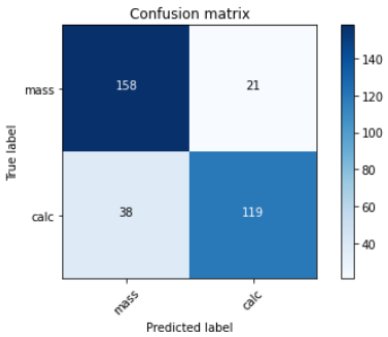


Figure 4: Confusion matrix for the test-set predictions

5.1.1 Data augmentation

To overcome the problem of over-fitting, we augmented our training using the *ImageDataGenerator* class of *keras*. The main variations we adopted were random rotation up to 40 degrees, random vertical and horizontal shifts with a range of 0.2, random zoom with a range of 0.2, random shear with a range of 0.2 and nearest fill model. To set these parameters we referred to the data in the papers we studied in the initial phase of the project. Before applying these variations to the train set, we splitted it to get the validation set separately using a validation size of the 15%.

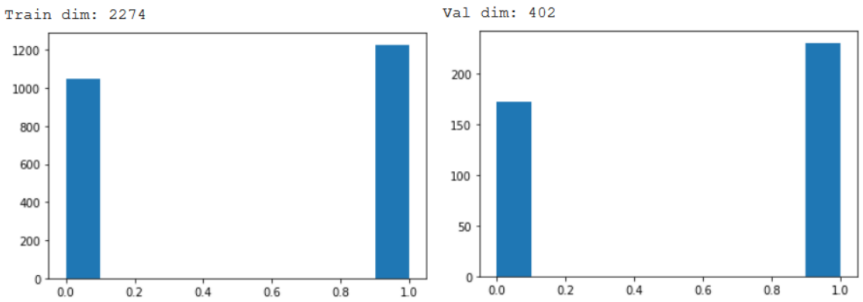


Figure 5: Label distribution of the validation set and the training set, after the split

We trained again our model with the augmented training set and with the validation set, and the results were more satisfying, having a mean accuracy over 80%. At this point, we wanted to have a more accurate analysis on some hyperparameters so we did several tests changing their values to get a model as good as possible. The tests were regarding the value of the *batch size*, the number of *hidden units*, the *learning rate* and the *optimizer* itself: we tried using the Adam optimizer and the RMSprop optimizer. Moreover, we also tested the model with different *output functions*: the softmax and the sigmoid. The final accuracy we managed to achieve was 87%. This result has been obtained with the Adam optimizer with a learning rate of 0.001 and with the $softmax$ as the output function for the last FC layer. Regarding to the number of hidden units of the first FC layer, the differences in terms of performance were limited (1 or 2 percentage points) therefore we have chosen the smaller value (64), so to have a model as simple as possible. The batch size was chosen at 50: we noticed that the 25 batch size models returned more unstable and variable results. The architecture of the trained network is the same as in figure 2, while the results in terms of accuracy and confusion matrix of the test set are shown in figures 6 and 7. All the tests were done with a maximum of 100 epochs, using the early stopping with a patience of 50.

Batch size	Hidden units	Test accuracy	Test loss	AUC	F-score
25	64	0.87	0.30	0.87	0.86
50	64	0.86	0.32	0.85	0.86
25	128	0.80	0.46	0.80	0.78
50	128	0.84	0.33	0.83	0.83
25	256	0.85	0.33	0.84	0.83
50	256	0.86	0.29	0.86	0.85
25	512	0.88	0.29	0.88	0.87
50	512	0.87	0.30	0.87	0.85

Table 1: Results of tests with Adam optimizer and softmax output function (*Task* 2.1)

Before saving the model, we performed a 4-holdout validation analysis to make sure that the performance remained similar to the values obtained in the previous test. This analysis confirmed the goodness of the model: the results of the 4-holdout validation are shown below.

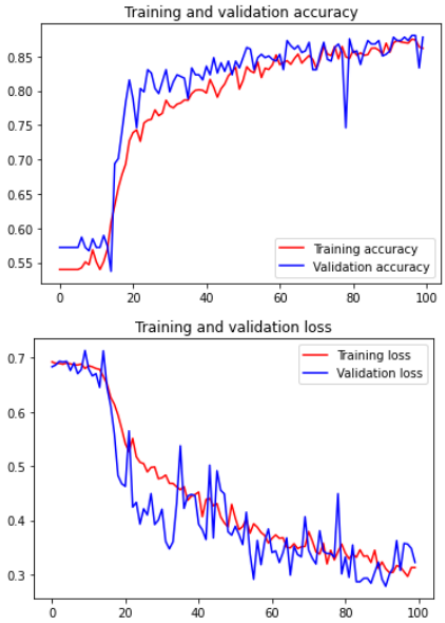


Figure 6: Accuracy and loss graphs for the model with data augmentation

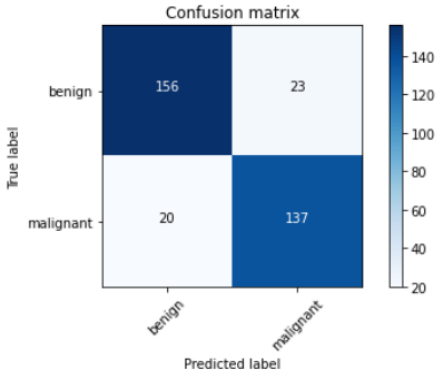


Figure 7: Confusion matrix for the test-set predictions

Batch size	Hidden units	Measure	1st	2nd	3rd	4th	**AVG**
50	**64**	accuracy	0.87	0.87	0.85	0.85	**0.86**
"	"	loss	0.30	0.30	0.30	0.32	**0.31**
"	"	AUC	0.87	0.87	0.85	0.85	**0.86**
"	"	F-score	0.86	0.86	0.84	0.83	**0.85**

Table 2: Results of 4-holdout validation (*Task* 2.1)

5.2 Task 2.2: Benign-Malignant discriminator

This classification was more complex to accomplish, and the results obtained are not comparable with previous experiments. One of the problems was that the distribution of the two classes was imbalanced in the training set. But the main problem was that distinguishing an abnormality patch between malignant and benign is generally more complex than distinguishing the type of abnormality. We spoke with a radiologist and some of his colleagues, and they explained to us that a calcification is very different to a mass: the first is always a very small abnormality, while the second one is almost always bigger. Moreover, they said that mammograms are considered first level examinations, which therefore can leave many doubts about the diagnosis. This is because there are many factors that can limit the visibility and the study of the abnormalities. In most cases, abnormalities are best investigated with subsequent examinations, such as ultrasound, MRI, and biopsy. This complexity remains even in deep learning techniques.

As a first approach, we started from the network built in the previous task and we added some modifications, based also on the feedback found on the papers. We used again the data augmentation, but modifying some of the parameters: in more than one paper it was specified that, to avoid too many distortions in the images, it was necessary to specify a rotation angle multiple of the right angle, and a fill mode of type 'reflect'. Moreover we disabled the horizontal flip of the image, to avoid too much differences from the original set. We inserted a dropout layer between the two dense layers at the top of the model, with a rate varying from 0.2 to 0.5 in the tests. The better results that we obtained are shown in figures 8 and 9: the used batch size was 50, the hidden units of the first dense layer were 256, and the optimizer was *Adam* with a learning rate of 0.001. This model was built with a rate of the dropout layer of 0.2.

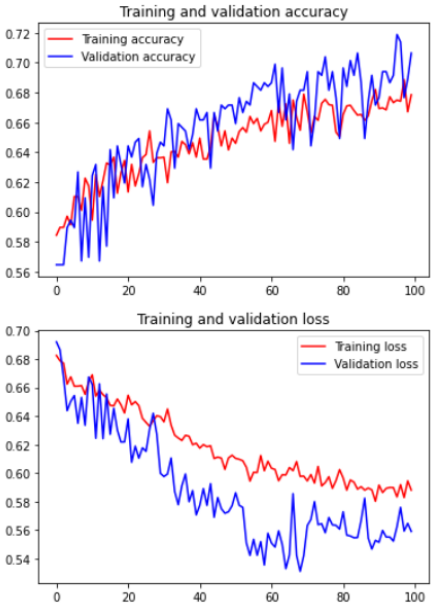


Figure 8: Accuracy and loss graphs for the first *Task* 2.2 model

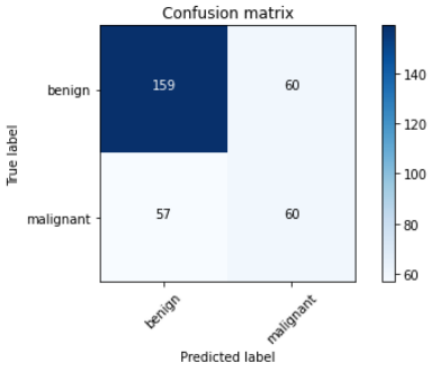


Figure 9: Confusion matrix for the test-set predictions

As we can see from the graphs, there are some improvement during the training of the network, but they're very limited and slow. We tried to increment the number of training epochs and the model started to overfitting badly. The major problem is that, predicting the test-set labels, the classifier tended to assign samples to the benign class, when we would have preferred more attention to the malignant class instead, since it is the most critical.

We tried to change the architecture of the model again, looking for a structure that could better capture the features of our dataset. After few attempts, we arrived at the model described in figure 10, in which we inserted an extra convolutional layer with 128 kernels. We maintained the number of hidden units of the FC layer to 64. This last choice has been made also to simplify the network, decreasing the number of parameters to train in order to reduce the overfitting. The dropout layer remained with a rate of 0.2.

Layer (type)	Output Shape	Param #
conv2d_18 (Conv2D)	(None, 148, 148, 32)	320
max_pooling2d_12 (MaxPooling	(None, 74, 74, 32)	0
conv2d_19 (Conv2D)	(None, 72, 72, 32)	9248
max_pooling2d_13 (MaxPooling	(None, 36, 36, 32)	0
conv2d_20 (Conv2D)	(None, 34, 34, 64)	18496
max_pooling2d_14 (MaxPooling	(None, 17, 17, 64)	0
conv2d_21 (Conv2D)	(None, 15, 15, 64)	36928
max_pooling2d_15 (MaxPooling	(None, 7, 7, 64)	0
conv2d_22 (Conv2D)	(None, 5, 5, 128)	73856
conv2d_23 (Conv2D)	(None, 3, 3, 128)	147584
flatten_3 (Flatten)	(None, 1152)	0
dense_6 (Dense)	(None, 64)	73792
dropout_3 (Dropout)	(None, 64)	0
dense_7 (Dense)	(None, 2)	130

Total params: 360,354
Trainable params: 360,354
Non-trainable params: 0

Figure 10: Final architecture of the CNN from scratch (*Task* 2.2)

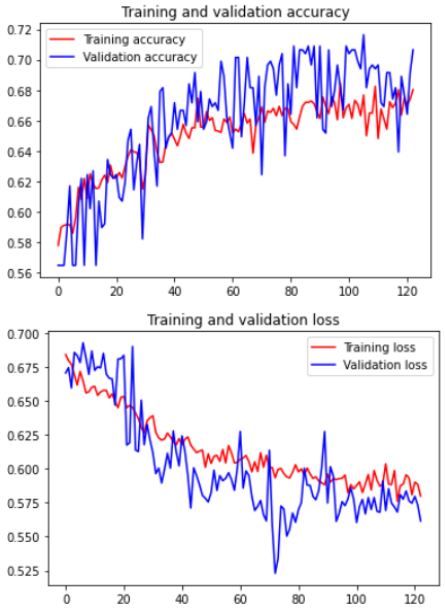


Figure 11: Accuracy and loss graphs for the *Task* 2.2 model with additional convolutional layer

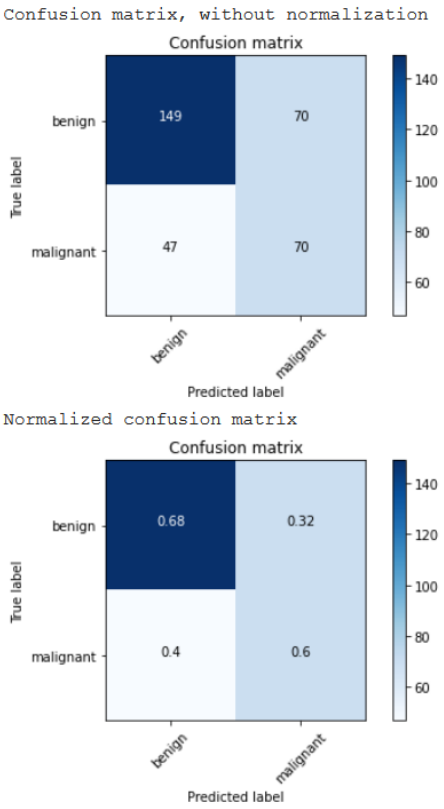


Figure 12: Confusion matrix for the test-set predictions

Batch size	Hidden units	Measure	1st	2nd	3rd	4th	**Avg**
50	**64**	accuracy	0.64	0.66	0.67	0.65	**0.66**
"	"	loss	0.64	0.61	0.60	0.63	**0.62**
"	"	AUC	0.64	0.63	0.64	0.64	**0.64**
"	"	F-score	0.56	0.53	0.53	0.54	**0.54**

Table 3: Results of 4-holdout validation (*Task* 2.2)

The results (fig. 11 and 12) in terms of accuracy and loss were very similar to the previous model, but we can see from the confusion matrix that the *malignant* class was better predicted and we considered it as a good result. As we can see from the table, the main problem is the value of the *F-score*, that indicates a very unbalanced classification.

Trying to solve the unbalancing of the two classes, we assigned weights to them so that their unbalance in the training set would be considered. We have trained again the network, passing the new weights of the classes as parameters: the results are found in the figures 13 and 14.

```
weight_for_benign = (1 / benign)*(total)/2.0
weight_for_malignant = (1 / malignant)*(total)/2.0

> Total samples: 2274
> Benign samples: 1341
> Malignant samples: 933
> Weight for benign: 0.85
> Weight for malignant: 1.22
```

We can see that using the class weights the classifier managed to find a slightly higher number of malignant samples, at the expense of benign samples. Comparing the holdout validation tables (3 and 4) we can see that accuracy and loss worsened, while the AUC remained almost unchanged, and the F-score increased by two percentage points. We also tried to change the values of the weights to see if you could improve performance further: our idea was to give even more weight to the malignant class, that was considered the most critical. But this approach did not satisfy us, because the network started to give too much importance to the class with the highest weight, and even most benign samples were classified as malignant.

So our choice for the final model was the version without the balanced class weights: although the F-score was higher in the one with the class weights, it paid with too much decrease of accuracy and too much increase of loss.

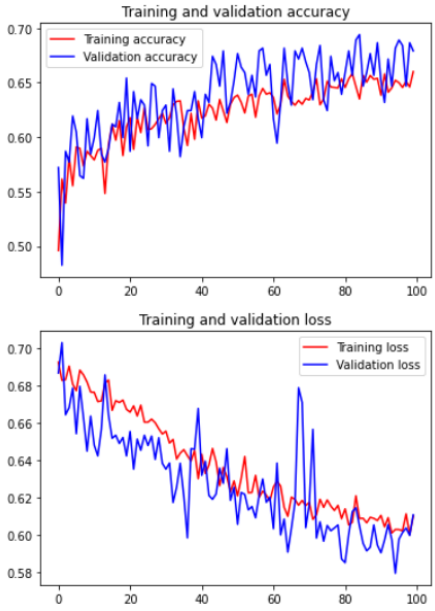


Figure 13: Accuracy and loss graphs for the *Task* 2.2 model with balanced class weights

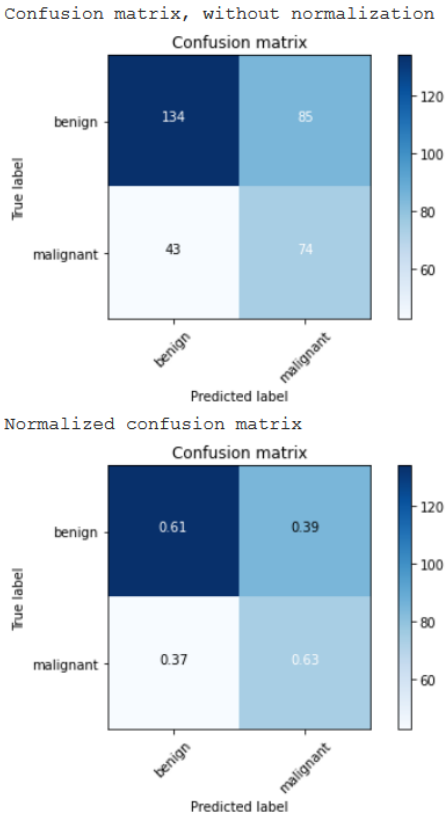


Figure 14: Confusion matrix for the test-set predictions

Batch size	Hidden units	Measure	1st	2nd	3rd	4th	**Avg**
50	**64**	accuracy	0.59	0.54	0.61	0.62	**0.59**
"	"	loss	0.66	0.71	0.64	0.62	**0.66**
"	"	AUC	0.64	0.61	0.64	0.62	**0.63**
"	"	F-score	0.57	0.56	0.57	0.54	**0.56**

Table 4: Results of 4-holdout validation with the class weights (*Task* 2.2)

6 Task 3: Pretrained network

For this task, we were inspired by some papers of works on the same dataset. Indeed, in most of the papers, pretrained networks were used to solve the classification. We found a study on the most used networks in the literature (shown in fig. 15) and then chose accordingly. The pretrained networks we worked on were:

- VGG-16
- VGG-19
- ResNet-50

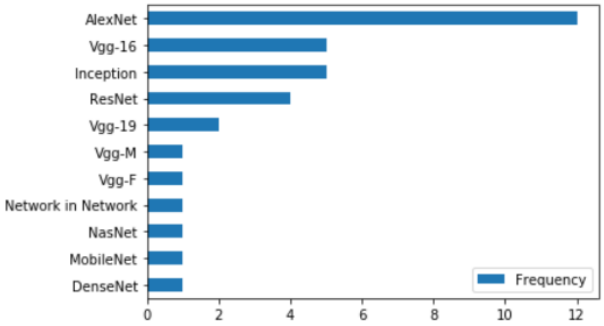


Figure 15: Pretrained networks most used in the literature

We used two strategies to leverage a pre-trained network. In the first one, called Feature Extraction, we started from a network trained on the "imagenet" dataset and removed the top layers and trained a new classifier on top of the output. The second one, called Fine Tuning is similar. The main difference was that the last layer of the model base used for feature extraction was unfrozen and we trained both the newly added part of the model and these top layers.
The results obtained with *Resnet-50* were not reported in this paper, because they weren't significant.

6.1 Task 3.1: Mass-Calcification discriminator

In all our tests we applied Data Augmentation because we ran into an overfitting problem. We built a new larger training set, starting from the original images and adding two more variations for each image. Thanks to this approach, we obtained improved results, in particular overtraining has been significantly reduced. The batch size value range that achieved higher performance is between 32 and 64. Lower values led to high oscillations of both the training and validation loss/accuracy. For the learning rate, after testing values from 1e-4 to 1e-6, we ended up using the value of 1e-5. After testing different activation functions we ended up using "sigmoid" for the output.

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
dense_1 (Dense)	(None, 2)	514

Total params: 16,812,610
Trainable params: 2,097,922
Non-trainable params: 14,714,688

Figure 16: Architecture of the CNN based on VGG16 (*Task* 3.1)

Layer (type)	Output Shape	Param #
vgg19 (Model)	(None, 4, 4, 512)	20024384
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 256)	2097408
dense_1 (Dense)	(None, 2)	514

Total params: 22,122,306
Trainable params: 2,097,922
Non-trainable params: 20,024,384

Figure 17: Architecture of the CNN based on VGG19 (*Task* 3.1)

6.1.1 Feature Extraction

We have reported the final architecture built from the **VGG16** base in figure 16. The best results were obtained with the following parameters: batch size 32, learning rate 1e-5.
Also from the **VGG19** convolutional base, whose architecture can be found in figure 17, we achieved the best results with a batch size of 32 and a learning rate of 1e-5.
The significant results achieved are summarized in the figures 18 and 19, and in the table 5.

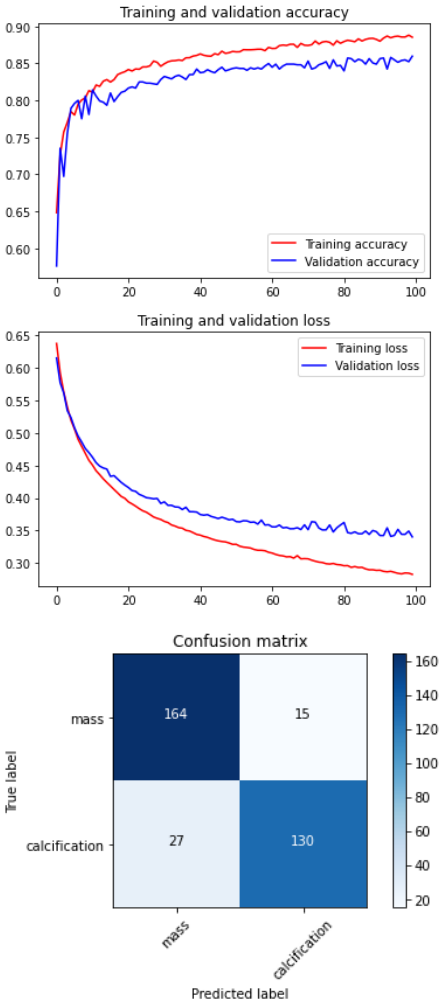


Figure 18: Performance for the model based on VGG16, with feature extraction

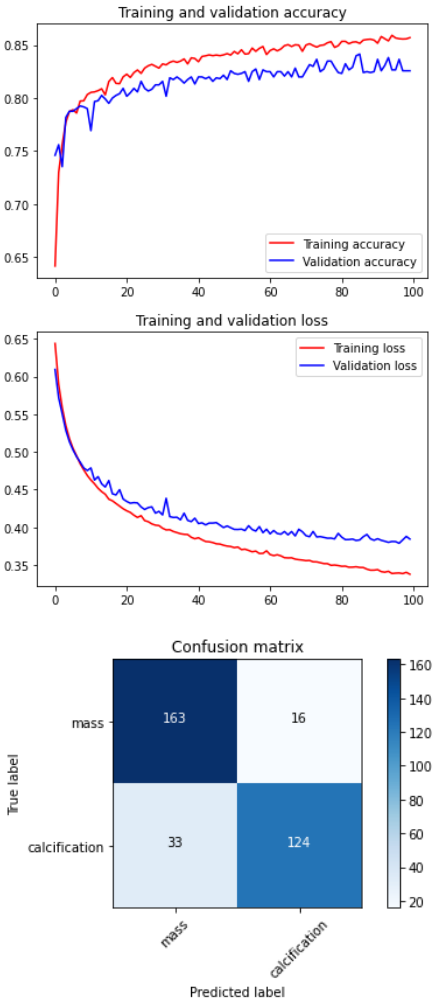


Figure 19: Performance for the model based on VGG19, with feature extraction

Model	Batch size	Test Acc	Test Loss	AUC	F-score
VGG16	32	**0.88**	**0.33**	**0.87**	**0.86**
"	64	0.86	0.34	0.86	0.85
VGG19	32	0.85	0.37	0.85	0.83
"	64	0.84	0.37	0.84	0.83

Table 5: Results of Feature Extraction (*Task* 3.1)

6.1.2 Fine Tuning

We also ran tests using the fine-tuning approach to see if we could increase classification performance.

As for VGG16, we started from the architecture in figure 16 and we carried out two different runs: the first unfreezing the pretrained network from the 5th block, while in the second the network is unfreezed from the 4th block. The results we obtained in these two cases were very similar, leading us to think that between the two would be better the first model, since it had less parameters to train.

Since the network tended to overtrain, we tried adding a dropout level between the two FC levels, with a rate of 0.5. The resulting outputs showed a decrease in overtraining, as we can observe in figures 20 and 21.

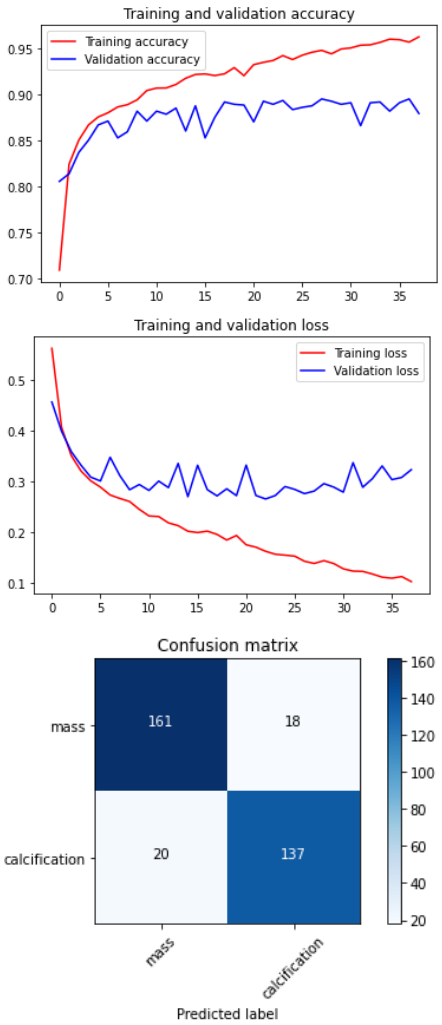


Figure 20: Performance for the model based on VGG16, with fine tuning

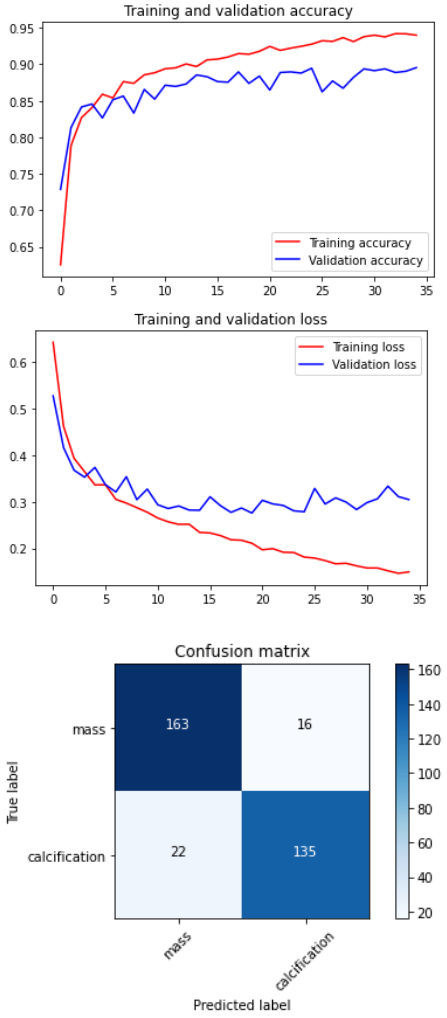


Figure 21: Performance for the model based on VGG16, with FT and dropout

Finally we re-trained VGG19, unfreezed from the 5th block: again performance was good, both with and without the dropout level. We reported a summary of all the obtained performances in the table 6.

For the final model, we saved the architecture based on VGG16, unfreezed from block 5, with the dropout level.

Model	Unfreezed block	Dropout	Test Acc	Test Loss	AUC	F-score
VGG16	5	-	0.89	0.34	0.89	0.88
VGG16	**5**	**0.5**	**0.89**	**0.33**	**0.89**	**0.88**
VGG16	4	-	0.88	0.32	0.88	0.88
VGG16	4	0.5	0.89	0.31	0.89	0.88
VGG19	5	-	0.85	0.36	0.85	0.84
VGG19	5	0.5	0.85	0.36	0.85	0.84

Table 6: Results of Fine Tuning (*Task* 3.1)

6.2 Task 3.2: Benign-Malignant discriminator

Regarding the classification of benign and malignant abnormalities, we used the same parameters but initially the classifier failed to properly classify malignant samples. To overcome this issue we decided to modify the weight for each class: at first we set the class weights to balance the dataset, but this wasn't improving the performance of the network. So we decided to set them heuristically giving more importance to the malignant class.

6.2.1 Feature Extraction

Using the **VGG16** convolutional base, the best results were obtained with the following parameters: batch size 64, learning rate 1e-5. We trained the model for 100 epochs, using an Early Stop callback with a patience of 10. These same parameters were also used for the tests with the **VGG19** base, and then for the tests with the class weights. The results are summarized in the table 7.
Using both the VGG16 and VGG19 models, we set the weight for the benign class at 1.0, while the one for the malignant class was set to 1.25.

Model	Weights	Test Acc	Test Loss	AUC	F-score
VGG16	-	0.66	0.59	0.63	0.52
VGG16	**[1.0,1.25]**	**0.68**	**0.59**	**0.64**	**0.54**
VGG19	-	0.67	0.61	0.62	0.48
VGG19	[1.0,1.25]	0.67	0.61	0.62	0.48

Table 7: Results of Feature Extraction (*Task* 3.2)

6.2.2 Fine Tuning

Since the obtained results were not very good, we decided to also try the fine tuning approach. Again, we ran several tests: initially we found an overtraining problem, and for this reason, we tried to insert a dropout level before the final dense level. The dropout rate was set to 0.5. The performance improved a bit. The table 8 shows the most significant test results.

Model	Unfreezed block	Dropout	Weights	Test Acc	Test Loss	AUC	F-score
VGG16	5	0.5	[1.0,1.25]	0.66	0.60	0.61	0.47
VGG16	5	0.5	-	0.68	0.60	0.61	0.47
VGG19	**5**	**0.5**	**[1.0,1.25]**	**0.65**	**0.62**	**0.63**	**0.53**
VGG19	5	0.5	-	0.66	0.62	0.63	0.52

Table 8: Results of Fine Tuning (*Task* 3.2)

As we can observe from the previous summary tables, with regard to the technique of feature extraction the model that has resulted better is the one based on VGG16 with the class weights, while for the fine tuning the best one is based on VGG19, again with the class weights. The graphs of accuracy and loss trends, together with the confusion matrices of these two models, are found in the figures 22 and 23. Between the two, as final model, we chose the VGG16-based, because in addition to have better performance it presented less overfitting.

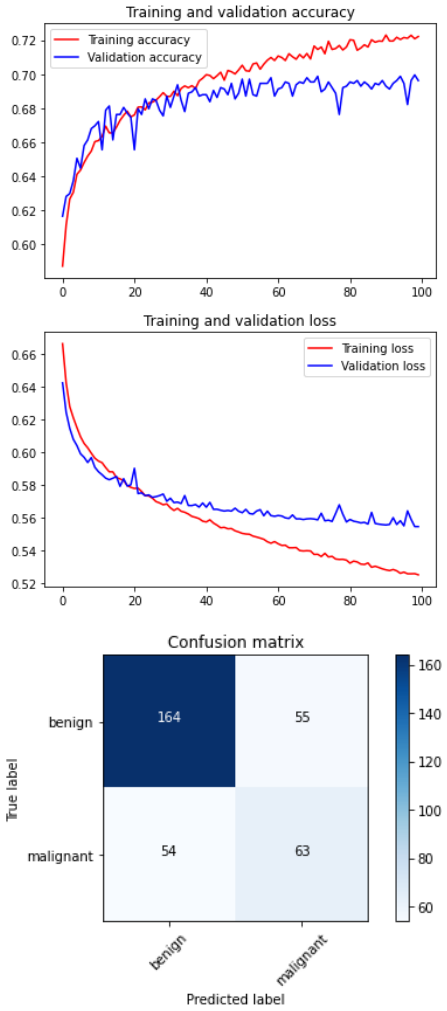


Figure 22: Performance for the **VGG16** based model, with FT and class weights

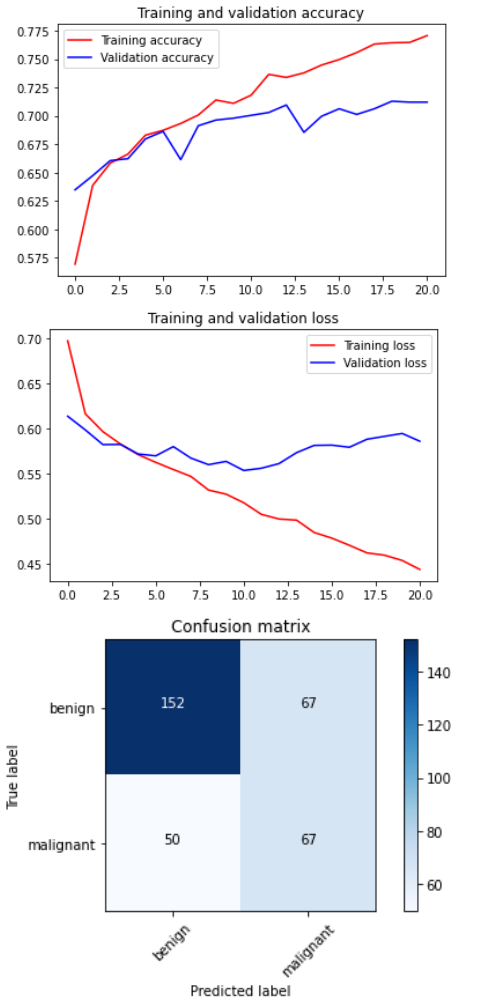


Figure 23: Performance for the **VGG19** based model, with FT and class weights

7 Task 4: Baseline patches

For this part of the project, we planned to integrate the baseline patches, the ones extracted from healthy tissues in the mammograms, into the classifiers identified so far. Our goal was to test whether adding the baseline patch samples into the training set would help the model better identify and learn features related to the abnormalities. We focused on the classification between mass and calcification.
So we started from the model architecture built from scratch in the *Task* 2.1 (fig. 2), and changed the number of output units to 3. We carried out some tests, modifying the hyper-parameters, to see the effect that the baseline patches had on the network training.

We initially trained with the architecture as it was, using again a batch size of 50, the softmax as output function, and the Adam optimizer with the default learning rate. Moreover we have maintained the use of data augmentation. The obtained results are found in figures 24 and 25. We can see that the performance are good, around the 80% of accuracy and F-score, but not as good as the model witouth the baseline.

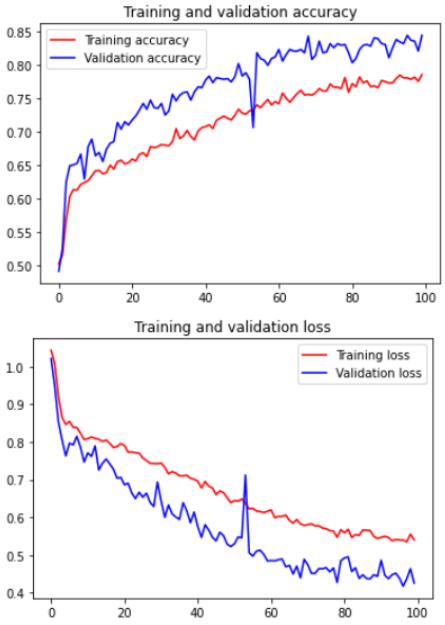


Figure 24: Accuracy and loss graphs for the model based on the CNN from scratch (*Task* 4)

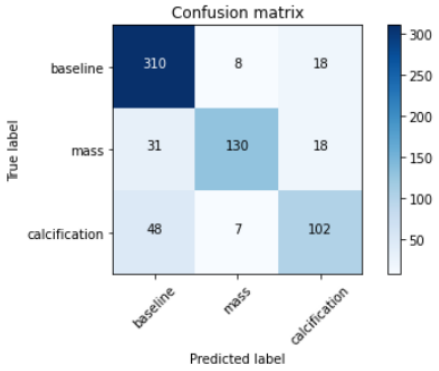


Figure 25: Confusion matrix for the test-set predictions

We thought this was due to the large imbalance in the dataset: in fact, the number of baseline patch samples was twice as large as the number of masses/calcifications. We therefore performed a subsequent test by rebalancing the class weights:

```
weight_for_baseline = (1 / baseline)*(total)/3.0
weight_for_mass = (1 / mass)*(total)/3.0
weight_for_calcification = (1 / calcification)*(total)/3.0

> Tot. samples: 4549
> Bas. samples: 2282, weight: 0.66
> Mass samples: 1041, weight: 1.46
> Calc samples: 1226, weight: 1.24
```

Here as in other tests, rebalancing the weights did not bring much improvement. However, we can see that mass and calcification classification improved, by 14% and 8% respectively, but at the expense of baseline classification which worsened by 16%. What we noticed in general, even in other tests that we did not report, is that calcifications are the abnormalities that are most often confused: in our opinion this is due to the fact that, as the radiologists with whom we were in contact had explained to us, calcifications are much smaller than masses, and consequently it could be more difficult to capture their features for neural networks, and confuse them with other classes.

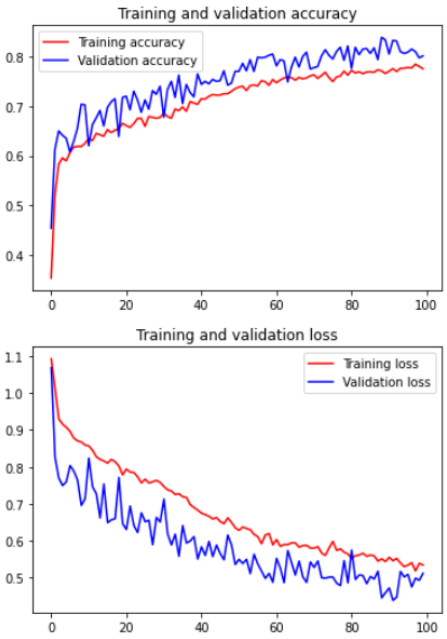


Figure 26: Accuracy and loss graphs for the model based on the CNN from scratch (*Task* 4)

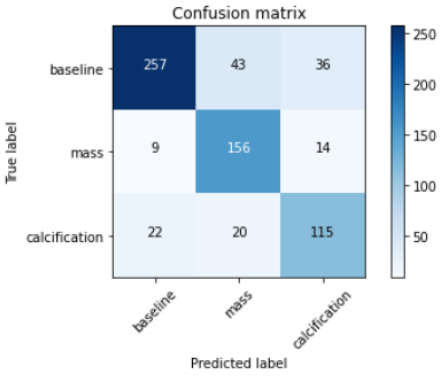


Figure 27: Confusion matrix for the test-set predictions

Model	Test Accuracy	Test Loss	F-score
batch:50, hidden u:64	0.83	0.46	0.80
" with weights	0.79	0.57	0.79

Table 9: Summary of the performance of the *Task* 4 tests

7.1 Siamese Network

Since the previous approach did not bring the expected results, we tried another approach, based on the architecture called Siamese Network. The idea was to send as input to the two identical networks of the model pairs of images, respectively to one a baseline patch and to the other an abnormality patch (mass or calcification).

At the output of the Flatten layer of the two common networks, we obtained the features extracted from the input and with a Lambda layer we performed the subtraction between the abnormality and baseline patch features. Our hope was that, from this difference, the most relevant features would be highlighted and the classification between masses and calcifications would improve. The final architecture we used is shown in figure 28: we started from the convolutional base of VGG16, keeping all blocks frozen. The we did an additional test, trying to unfreeze the convolutional base from the 5th block. The results of these tests are shown in the figures 31 and 32: note that the Early Stop callback was used with a patience of 10 and with the best weights restoring mode activated.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 150, 150, 3)	0	
input_3 (InputLayer)	[(None, 150, 150, 3)	0	
sequential (Sequential)	(None, 256)	16812096	input_2[0][0] input_3[0][0]
lambda (Lambda)	(None, 256)	0	sequential[1][0] sequential[2][0]
dense_1 (Dense)	(None, 2)	514	lambda[0][0]

Total params: 16,812,610
Trainable params: 2,097,922
Non-trainable params: 14,714,688

Figure 28: Architecture of the Siamese Network

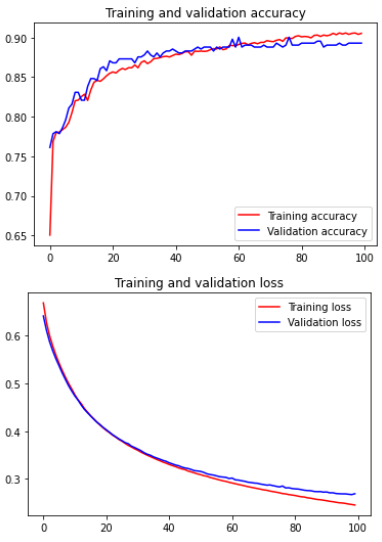


Figure 29: Accuracy and loss graphs for the Siamese Network (feature extraction)

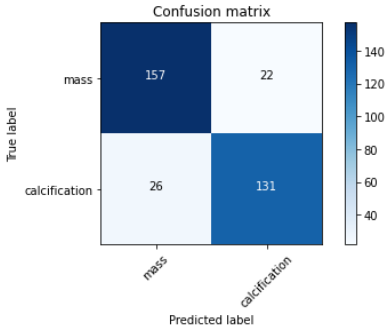


Figure 30: Confusion matrix for the test-set predictions

Model	Unfreezed block	Dropout	Test Accuracy	Test Loss	AUC	F-score
VGG16	-	-	0.86	0.34	0.86	0.85
"	5	-	0.87	0.33	0.87	0.86
"	5	0.5	0.87	0.34	0.87	0.86

Table 10: Summary of the performance of the Siamese Network tests

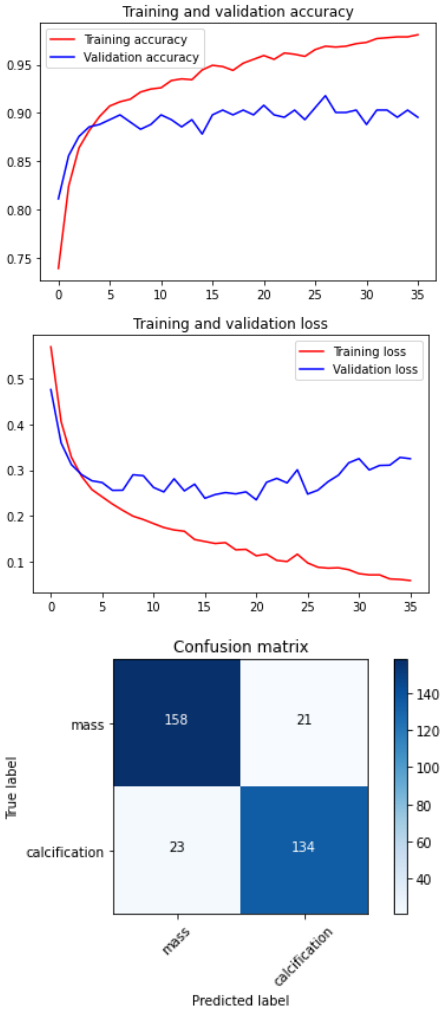


Figure 31: Performance for the Siamese Network, with FT

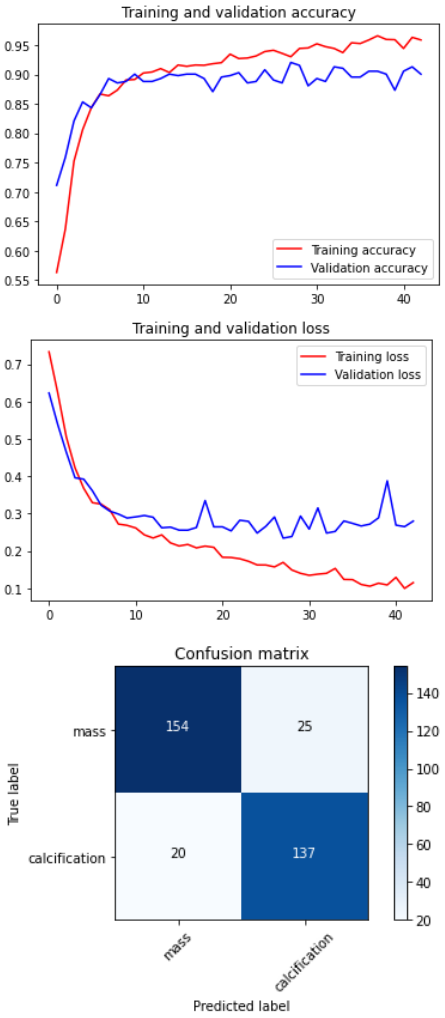


Figure 32: Performance for the Siamese Network, with FT and dropout

8 Task 5: Ensemble network