Contents

1 Introduction

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

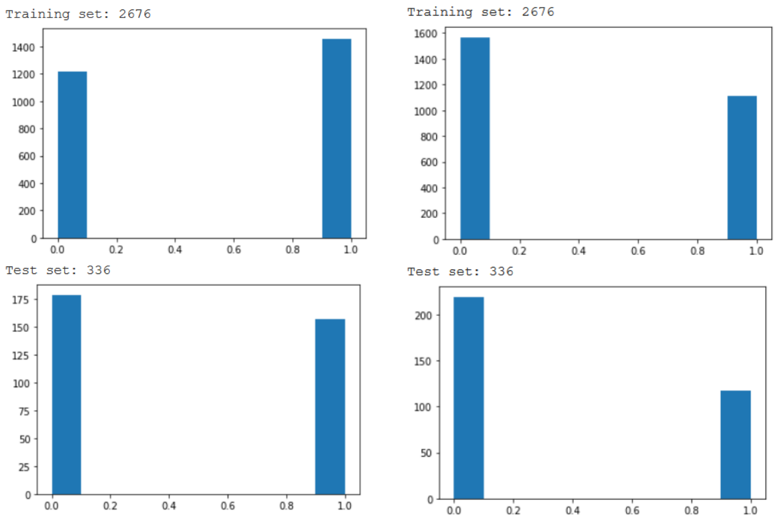
- Transfer Learning and Fine Tuning in Breast Mammogram Abnormalities Classification on CBIS-DDSM Database (Lenin G. Falconi, Maria Perez, Wilbert G. Aguilar, Aura Conci)
-
-
-
-

4 Task 2: From Scratch CNN

We developed different models of neural networks to work with the given dataset from scratch and we did different test to find the better hyperparameters to build the final model. This section is divided in three parts: in the first one we describe the main preprocessing applied to the data before the training of the models. Then we show how we built a model for classifying the dataset images between *mass* abnormality and *calfication* abnormality. In the last part, we describe the model built to classify between *benign* and *malignant* abnormality.

4.1 Data Preprocessing

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 al 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 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 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*

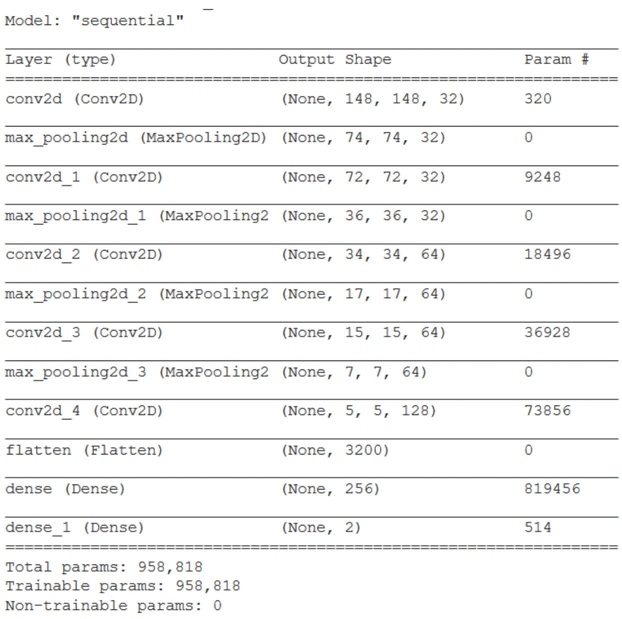
Figure 1: Label distribution

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. With regards to the labels, they has been categorizated so that is possible to use the categorical loss function in the model.

4.2 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 out 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* and different *output functions*. 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 preprocessing.



```
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, 256)               819456
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 514
=================================================================
Total params: 958,818
Trainable params: 958,818
Non-trainable params: 0
```

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

In the figure 3, we present the results in term of accuracy and loss, but also the consufion matrix (fig. 4), related to a test of the model with 256 hidden units, a batch size of 50, and the *softmax* function for the output layer. The used optimizer is Adam, with the deafult learning rate of 0.001. As we said before, the gap between the training and validation graphs show 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.

```
network.compile(optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy'])

network.fit(train_images, train_labels, epochs=100, batch_size=50,
        validation_split=0.15, shuffle=True, callbacks=[callback])
```

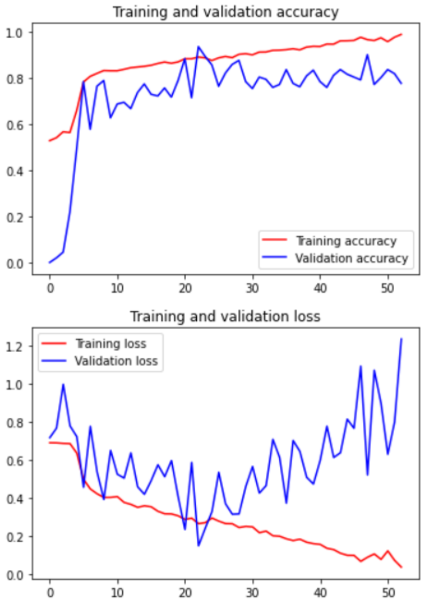


Figure 3: Accuracy and loss graphs for the first model training

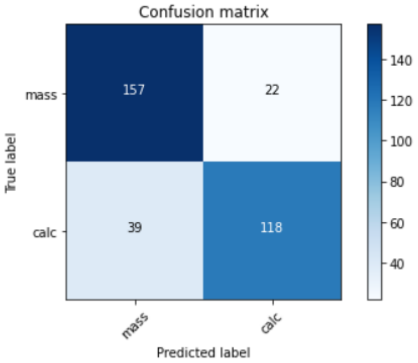


Figure 4: Confusion matrix for the test-set predictions

4.2.1 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.

```
train_set, val_set, train_classes, val_classes =
        train_test_split( train_images, train_labels, test_size=0.15)

train_datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=20,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode=’nearest’)

train_generator = train_datagen.flow(train_set,train_classes,batch_size=50)
```

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. The batch size was chosen at 50 and the number of hidden units for the forst FC layer at 256, both as in the first model without data augmentation. 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 5 and 6. All the tests were done with a maximum of 100 epochs, using the early stopping with a patience of 30.

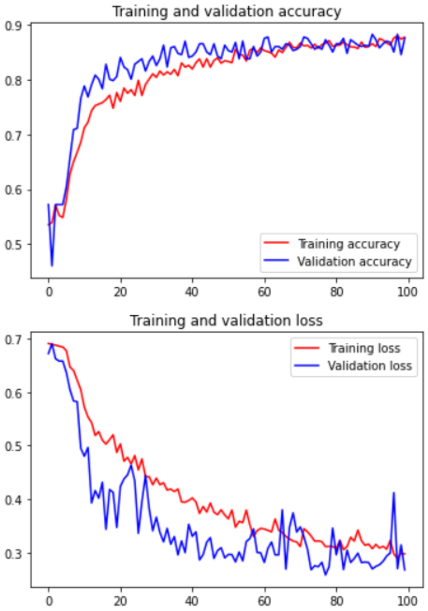


Figure 5: Accuracy and loss graphs for the model with data augmentation

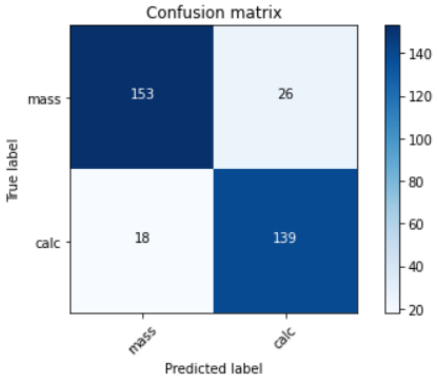


Figure 6: Confusion matrix for the test-set predictions

4.3 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 7 and 8: the used batch size was still 50, the hidden units of the first dense layer were 256, and the learning rate was 0.001. This model was built with a rate of the dropout layer of 0.2.

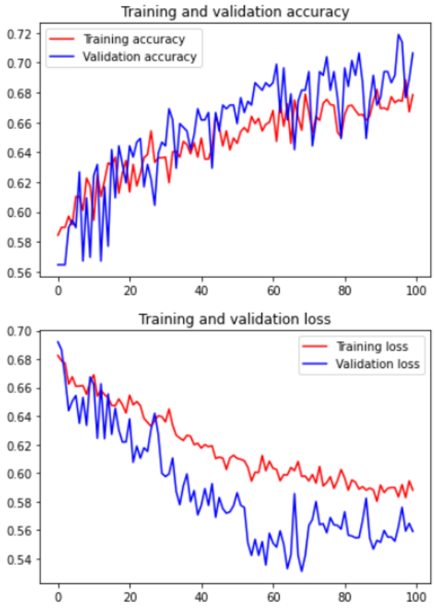


Figure 7: Accuracy and loss graphs for the model with data augmentation

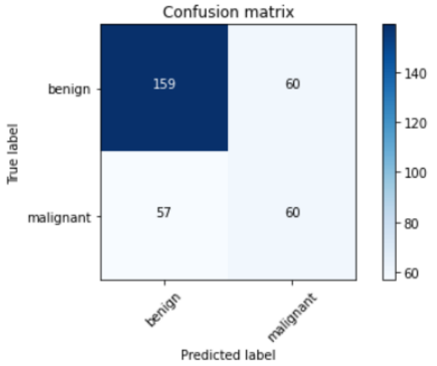
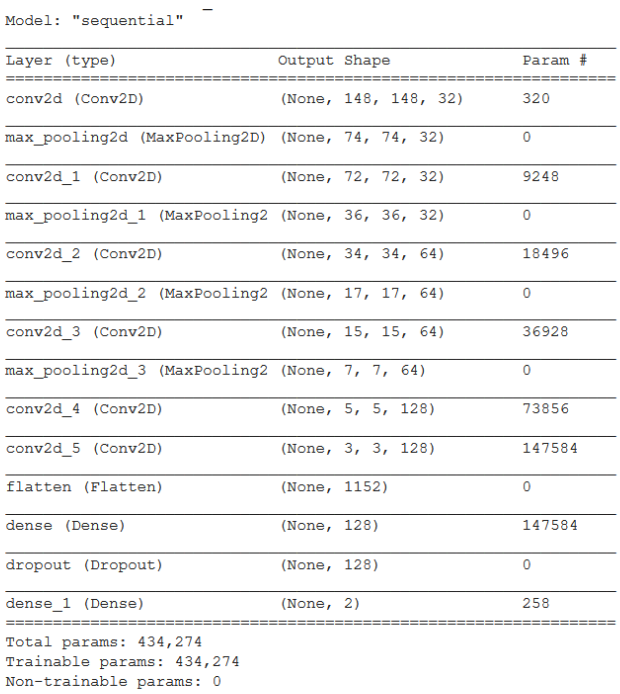


Figure 8: Confusion matrix for the test-set predictions

As we can see from the graphs, there are some improvement during the training of the network, but very limited. 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 9, in which we inserted an extra convolutional layer with 128 kernels, and we decreased the number of hidden units of the FC layer to 128. This last modification has been done 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.



```
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
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 128)         147584
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 128)               147584
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 434,274
Trainable params: 434,274
Non-trainable params: 0
```

Figure 9: Architecture of the CNN from scratch (*Task* 2.2)

The results (fig. 10 and 11) 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.

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 12 and 13.

```
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
```

Finally, we tried a different approach to see if performance could improve. We used the data augmentation functions to create a new training set with a larger number of samples: from each image we created four more, applying the *ImageDataGenerator* transformations, and obtaining a new dataset of 13.380 samples.

```
from keras.preprocessing.image import ImageDataGenerator
variations_per_img = 4
train_datagen = ImageDataGenerator(
    rotation_range=90,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=20,
    zoom_range=0.2,
    fill_mode='reflect')

for i in range(0,len(train_images)):
  new_train_images.append(train_images[i])
  new_train_labels.append(train_labels[i])
```

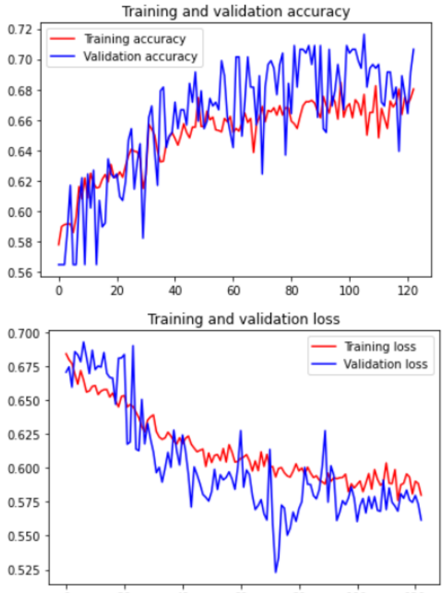


Figure 10: Accuracy and loss graphs for the model with data augmentation

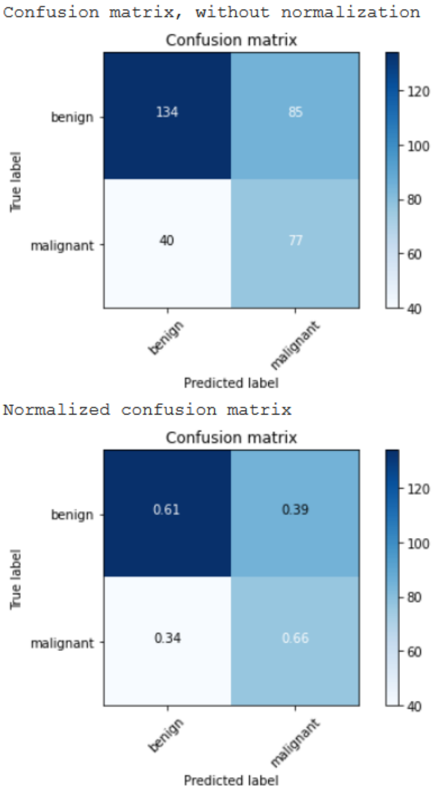


Figure 11: Confusion matrix for the test-set predictions

```
train_generator = train_datagen.flow(
  train_images[i].reshape(1,150,150,1),batch_size=1)
for j in range(variations_per_img):
  new_train_images.append(train_generator.next().reshape(150,150))
  new_train_labels.append(train_labels[i])
del train_generator
```

The classification did not improve, and indeed this model was experimenting a higher rate of over-fitting. We tried again to reduce the over-training, by adding some dropout layers, also between the convolutional layers, by further simplifying the model, and by decreasing the learning rate, but we did not succeeded.

4.3.1 Mass benign - mass malignant discriminator

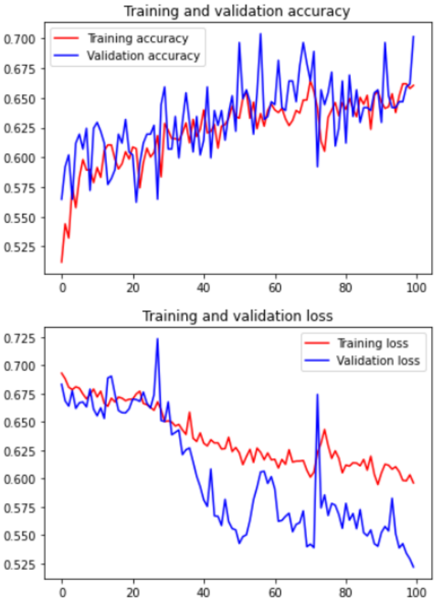


Figure 12: Accuracy and loss graphs for the model with data augmentation

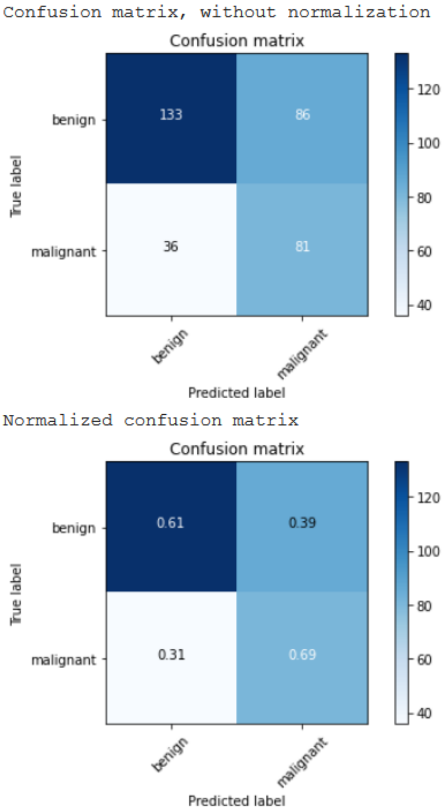


Figure 13: Confusion matrix for the test-set predictions

5 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. 14) and then chose accordingly. The pretrained networks we worked on were:

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

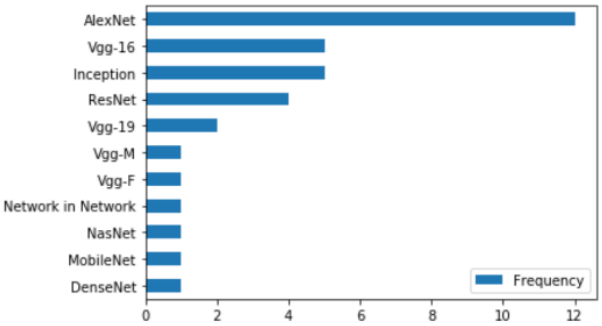


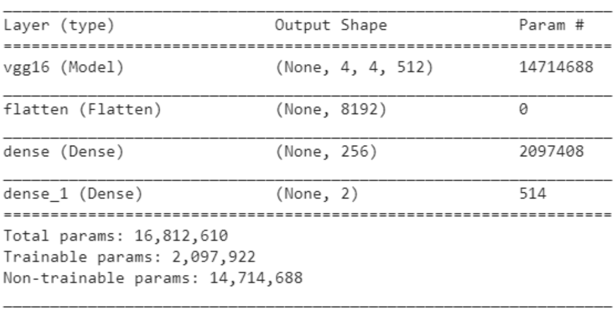
Figure 14: Pretrained networks most used in the literature

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 article, because they weren’t significant.

5.1 Task 3.1: Mass-Calcification discriminator

In all our tests we applied Data Augmentation because we ran into an overfitting problem. Two more variations for each image were added and 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.

The final architecture obtained that has been built using **VGG16** is shown in figure 18.



```
_________________________________________________________________
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 15: Architecture of the CNN based on VGG16 (*Task* 3.1)

Feature extraction:
The best results were obtained with the following parameters: batch size 50, learning rate 1e-5. We obtained an accuracy of XXX after training for 100 epochs.

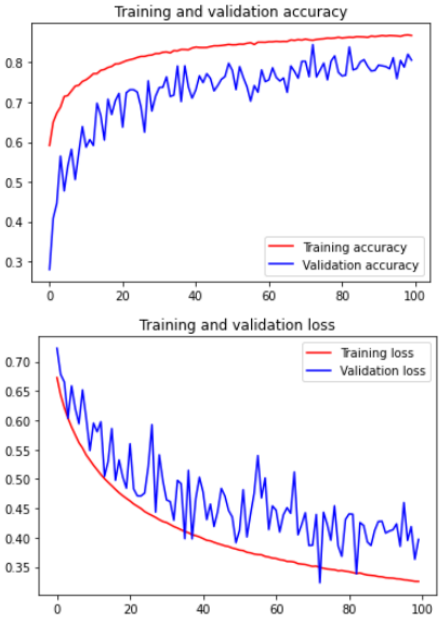


Figure 16: Accuracy and loss graphs for the model based on VGG16

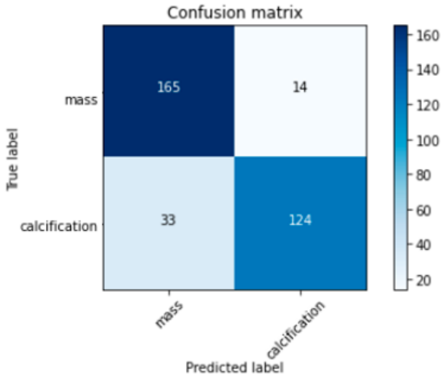
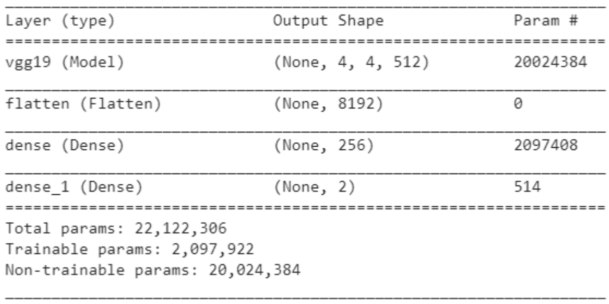


Figure 17: Confusion matrix for the test-set predictions

Fine Tuning:

VGG19: The final architecture that has been built using **VGG19** is shown in figure **??**.



```
_________________________________________________________________
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 18: Architecture of the CNN based on VGG19 (*Task* 3.1)

Feature extraction:
The best results were obtained with the following parameters: batch size 32, learning rate 1e-5. We obtained an accuracy of XXX after training for 65 epochs.

Fine Tuning:

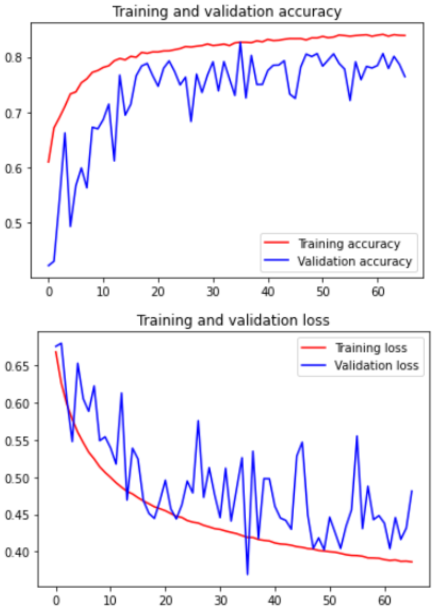
5.2 Task 3.2: Benign-Malignant discriminator

Regarding the classification of benign and malignant we used the same parameters but as we can see from the confusion matrix in figure 21 the classifier fails to properly classify malignant samples. To overcome this issue we decided to add a weight for each class, giving more importance to the malignant class.

Using the **VGG16** convolutional base, the best results were obtained with the following parameters: batch size 50, learning rate 1e-5. We obtained an accuracy of XXX after training for 100 epochs.

The class weights that we setted were:

```
class_weight = { 0: 1, 1: 1.25}
```



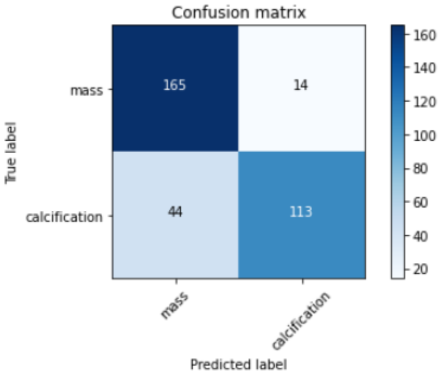


Figure 20: Confusion matrix for the test-set predictions

Figure 19: Accuracy and loss graphs for the model based on VGG19

Instead, using the VGG19 convolutional base, the best results were obtained with the following parameters: batch size 32, learning rate 1e-5. We obtained an accuracy of XXX after training for 65epochs.

The class weights that we setted were:

```
class_weight = { 0: 1, 1: 1.2}
```

6 Task 4: Baseline patches

7 Task 5: Ensemble network

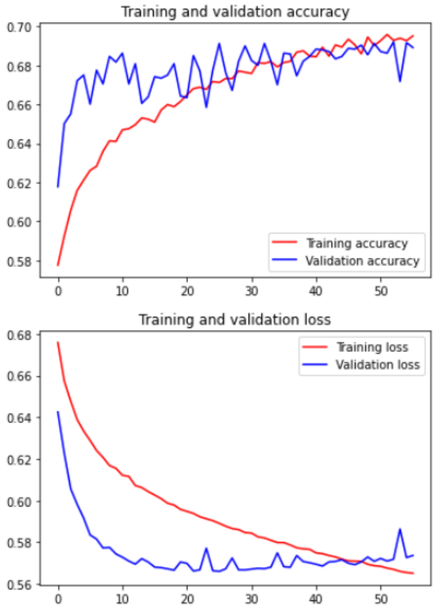


Figure 21: Accuracy and loss graphs for the model based on VGG16, without weights

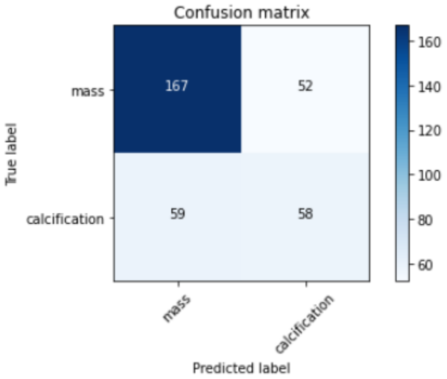


Figure 22: Confusion matrix for the test-set predictions, without weights

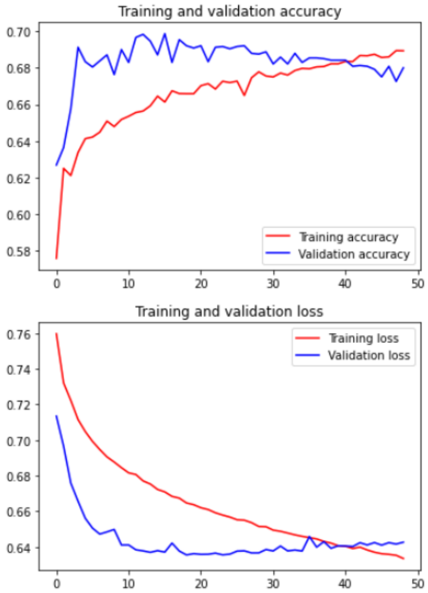


Figure 23: Accuracy and loss graphs for the model based on VGG16, with weights

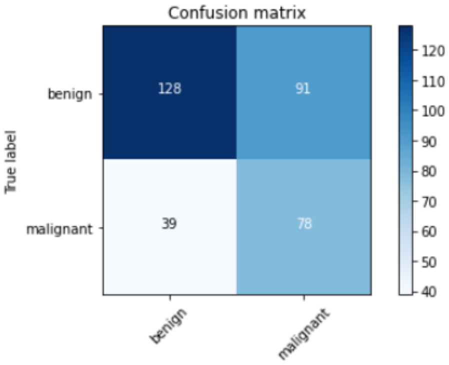


Figure 24: Confusion matrix for the test-set predictions, with weights

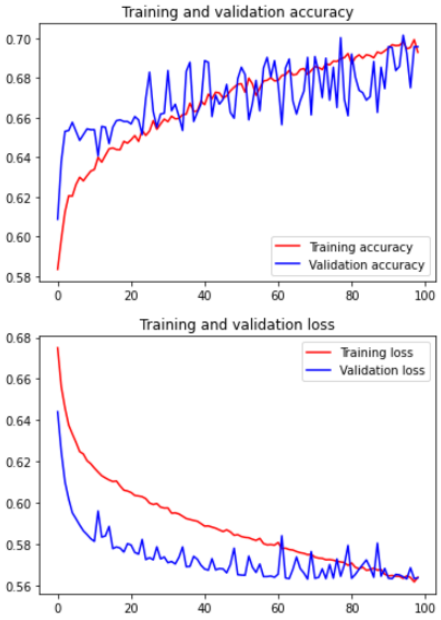


Figure 25: Accuracy and loss graphs for the model based on VGG19, without weights

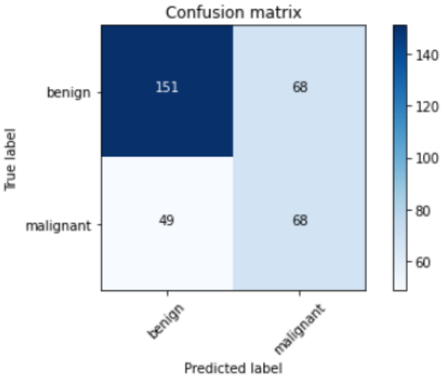


Figure 26: Confusion matrix for the test-set predictions, without weights

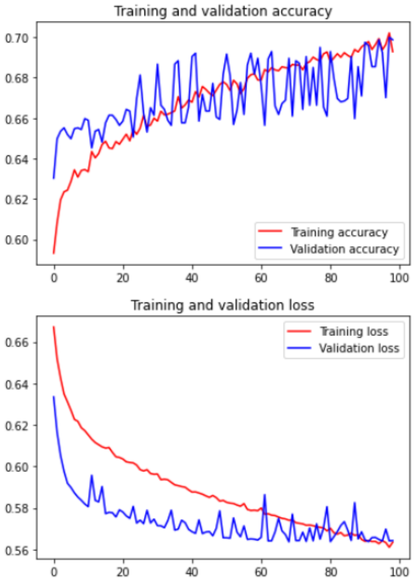


Figure 27: Accuracy and loss graphs for the model based on VGG19, with weights

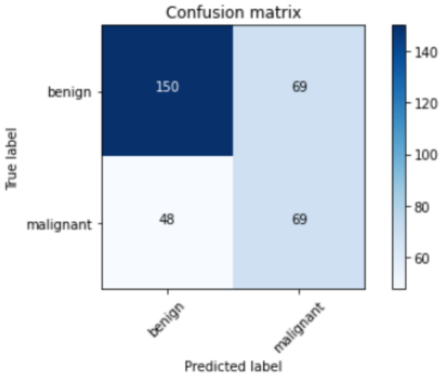


Figure 28: Confusion matrix for the test-set predictions, with weights