Documentation of Computational Intelligence and Deep Learning project

Marco Parola
Stefano Poleggi

A.Y. 2020-2021

Contents

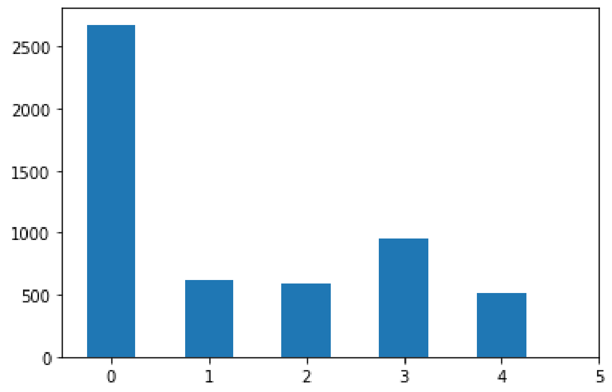
1 Introduction

1.1 Data distribution

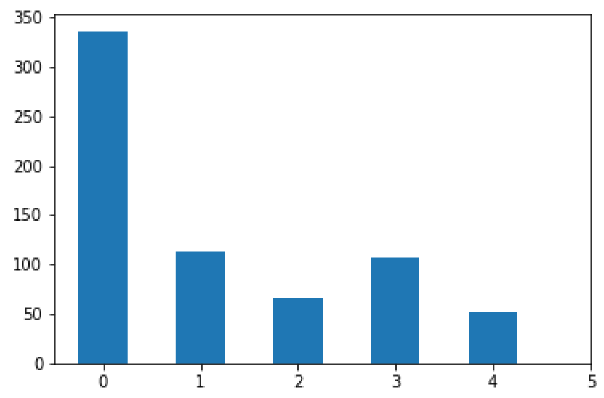
The dataset of this project has been created from the well known dataset CBIS DDSM (Curated Breast Imaging Subset of Digital Database for Screening Mammography), considering a sub number of radiography already labeled on the following classes:

- 1 Mass, Benign
- 2 Mass, Malignant
- 3 Calcification, Benign
- 4 Calcification, Malignant

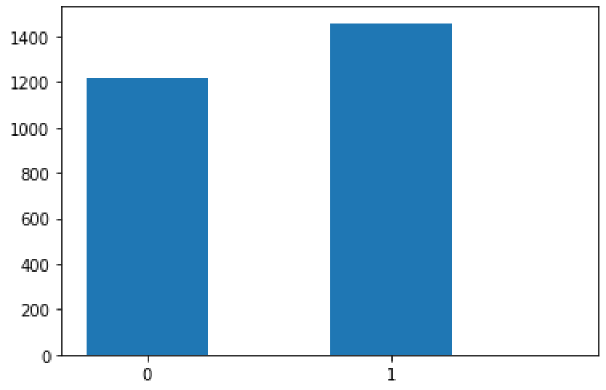
Moreover for each image belongs to the previous classes is present another image extracted from the same radiography, but representing a healthy portion.
The following plot show the number of sample for each class belongs to the trainingset.



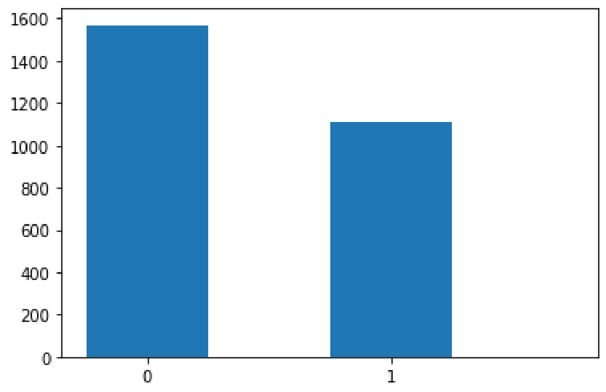
The following plot show the number of sample for each class belongs to the testset.



The following plot show the number of sample belonging to the class Mass and to the class Calcification.



The following plot show the number of sample belonging to the class Benign and to the class Malignant.



1.2 Benign-Malignant classification VS Mass-Calcification classification

During all the experiments we observed a substantial difference between the classification benign-malignant and the classification mass-calcification.
In our opinion this could be caused by a strong similarity between mass benign and mass malignant (same for calcification) with respect to benign mass and benign calcification (same for malignant).

1.3 Compute shuffle on the dataset

During the first experiments we observed some picks on the accuracy and loss plots, we solved this problem simply preprocessing the data, computing a shuffle phase, in which we computed a parmutation on the images and the same transofmation on the labels, in order to maintain the consistency.

```
training = []
for i in range(len(trainingSet)):
  training.append((trainingSet[i], trainingLabels[i]))

training = np.array(training)
np.random.shuffle(training)
```

2 Task 1

We read the following paper to study the approcches followed by some researches:

- Transfer Learning with Convolutional Neural Network for Early Gastric Cancer Classification on Magnifiying Narrow-Band Imaging Images, published in 2018 25th IEEE International Conference on Image Processing (ICIP).
- Multi-View Feature Fusion Based Four ViewsModel for Mammogram Classification UsingConvolutional Neural Network,published in 2019 26th November IEEE.
- Analysis on the Dropout Effect in Convolutional Neural Networks

Starting from these papers, we took ispiration in different aspects.

2.1 Pretrained Convolutional Neural Networks

Considering different experiments described on the journals, we detected some pretrained CNN most used for medical images classification:

- AlexNet
- VGG16
- GoogLeNet
- ResNet

We analyzed this topics more in detail on section related to the task 3.

2.2 Setting of Hyper-parameters

We took inspiration to set the hyper-parameters with wich we started training our models from the second and third journal.
In our experiments we used the same learning rate values, the activation functions and the number of convolutional layers (in the 'from Scratch' tasks), used in **Multi-View Feature Fusion Based Four ViewsModel for Mammogram Classification UsingConvolutional Neural Network** paper.
Moreover from this paper we took the hyper-parameters of the *ImageDataGenerator*, thanks wich we computed the data augmentation:

- Rotation = 90
- Horizontal shift range = 0.2
- Vertical shift range = 0.2

- Zoom range = 0.2
- Shear = 20
- Fill model = 'nearest'

On section related to the task 2.2 we will show the hyperparameters that we set, starting from these.
From the **Analysis on the Dropout Effect in Convolutional Neural Networks** journal we took inspiration about the dropout rate values, in particular we used the following guidelines:

- for dropout rate between convolutional layers we used a value between 0.1 and 0.2
- for dropout rate between dense layers we used a haigher value: 0.5

3 Task 2

Task 2 consists in building from scratch different convolotional neural networks able to compute the following classification:

- Mass - Calcification
- Benign - Malignant

We have carried out this task adopting an approach ”trial and error”, in this way we found a our best classifier step by step.

3.1 Task 2.1, Mass - Calcification

We compiled our deep nural networks using two optimizers:

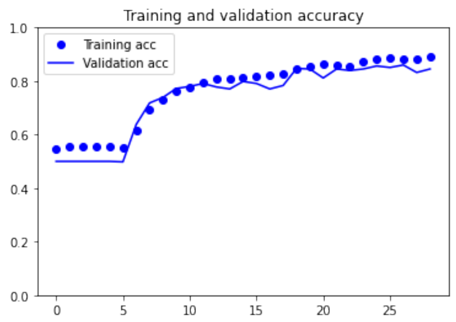
- adam
- RMSprop

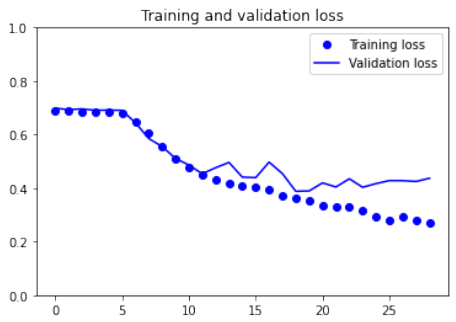
In all these experiments we used an early stopping condition, we set the *patience* value euristicly equal to 10.
Moreover exploiting the *restore_best_weights* itroduced in keras 2.2.3 we set the best weights of the model found previously.

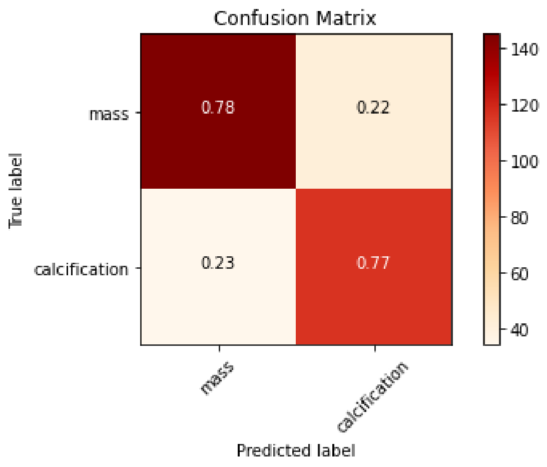
3.1.1 Model 1

- Optimizer: Adam, learning rate = 0.001 (default)
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 2
- Activation function: softmax

```
              precision    recall  f1-score   support

           0       0.80      0.81      0.81       179
           1       0.78      0.77      0.78       157

    accuracy                           0.79       336
   macro avg       0.79      0.79      0.79       336
weighted avg       0.79      0.79      0.79       336


0s 17ms/step - loss: 0.4683 - accuracy: 0.7887
```







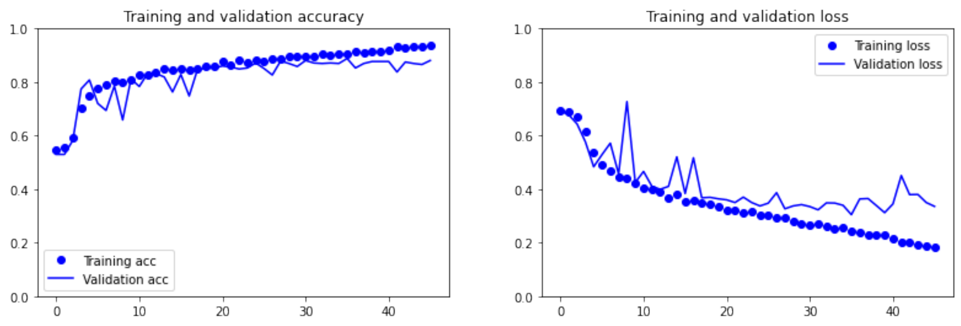
3.1.2 Model 2

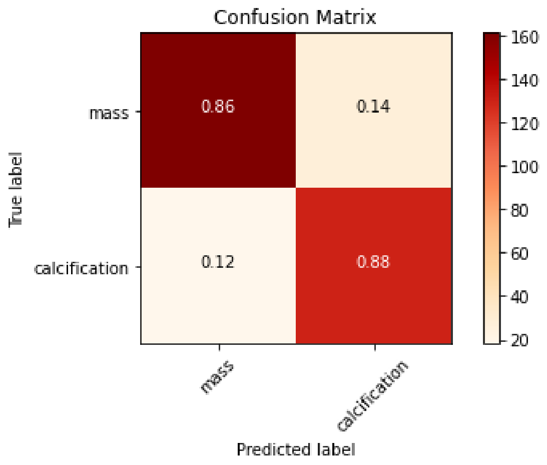
We modified the architecture of the network, adding one more convolutional layer before the first two max-pooling layers.

- Optimizer: RMSProp, learning rate = 1e-4
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1
- Activation function: sigmoid

```
          precision    recall  f1-score   support

0           0.84      0.88      0.86       179
1           0.86      0.82      0.84       157
```

```
    accuracy                         0.85       336
   macro avg       0.85      0.85      0.85       336
weighted avg       0.85      0.85      0.85       336

1s 44ms/step - loss: 0.3996 - accuracy: 0.8512
```





By applying this modification to the previous neural network, we can observe an improvement on the accuracy value of about 9% and a decrease on the loss of about 18%.

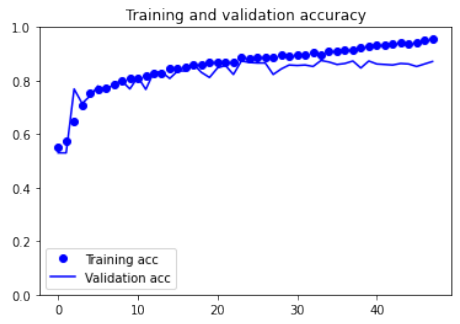
3.1.3 DropOut Model

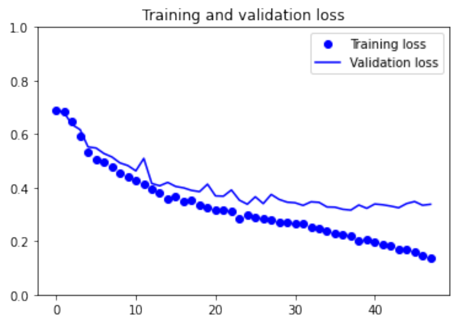
In the previous loss function plot we can observe a slight deviation between the loss function of the training set and validation set, that could imply our model is affected by overfitting. In order to solve this problem we introduce some dropout layers has seen in ”Analysis on the Dropout Effect in Convolutional Neural Networks” journal, with a percentage equal to 0,15.

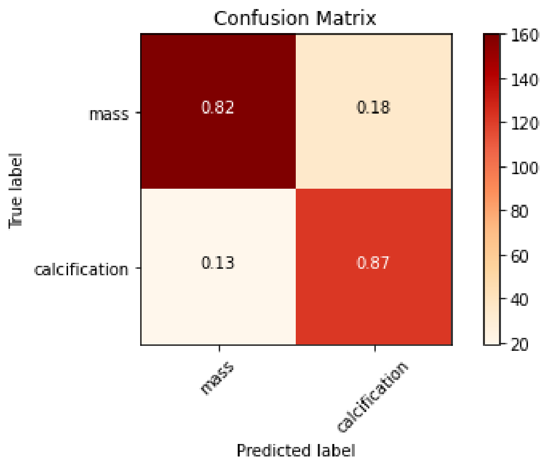
- Optimizer: RMSProp, learning rate = 1e-4
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1
- Activation function: sigmoid

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_35 (Conv2D)           (None, 150, 150, 32)      320
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 150, 150, 64)      18496
_________________________________________________________________
dropout_4 (Dropout)          (None, 150, 150, 64)      0
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 75, 75, 64)        0
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 75, 75, 64)        36928
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 75, 75, 64)        36928
_________________________________________________________________
dropout_5 (Dropout)          (None, 75, 75, 64)        0
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 37, 37, 128)       73856
_________________________________________________________________
dropout_6 (Dropout)          (None, 37, 37, 128)       0
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 18, 18, 128)       0
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 18, 18, 256)       295168
_________________________________________________________________
dropout_7 (Dropout)          (None, 18, 18, 256)       0
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 9, 9, 256)         0
_________________________________________________________________
flatten_6 (Flatten)          (None, 20736)             0
_________________________________________________________________
dense_14 (Dense)             (None, 256)               5308672
_________________________________________________________________
dense_15 (Dense)             (None, 2)                 514
=================================================================
Total params: 5,770,882
Trainable params: 5,770,882
Non-trainable params: 0
_________________________________________________________________
```

```
              precision    recall  f1-score   support

           0       0.81      0.91      0.85       179
           1       0.87      0.75      0.81       157

    accuracy                           0.83       336
   macro avg       0.84      0.83      0.83       336
weighted avg       0.84      0.83      0.83       336

1s 44ms/step - loss: 0.3807 - accuracy: 0.8333
```







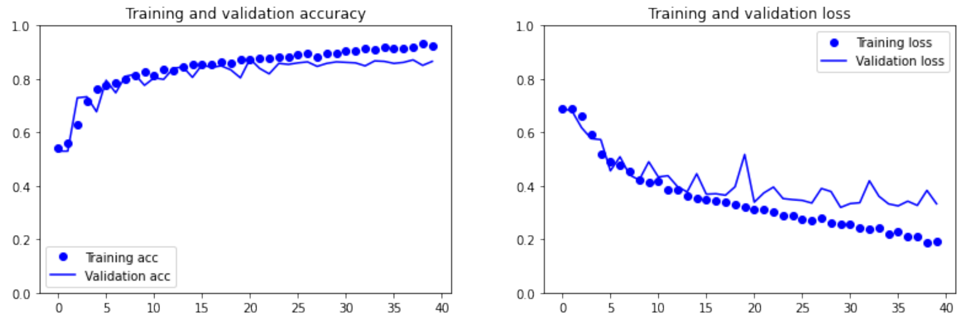
There is a bit decrese both on the loss and accurancy, by looking at the plot of loss value we can observe the validation curve is more regular than the previous experiments, solving partially the overfitting problem.

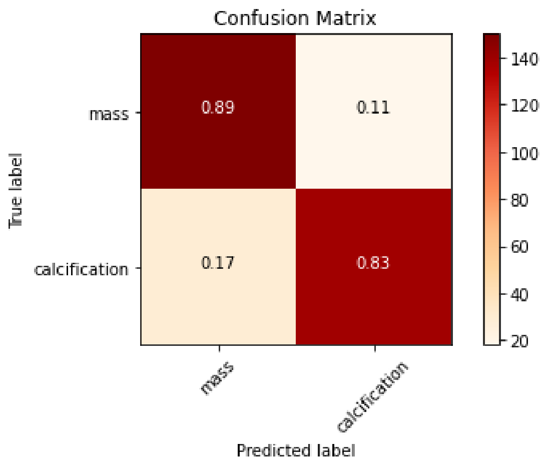
3.1.4 Model 4

We also try to improve the performance of our model, modifying the dense layers of the nettwork and adding one more hidden layer.

- Optimizer: RMSProp, learning rate = 1e-4
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 2
- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.80      0.92      0.86       179
           1       0.89      0.74      0.81       157

    accuracy                           0.84       336
   macro avg       0.85      0.83      0.83       336
weighted avg       0.84      0.84      0.83       336

1s 44ms/step - loss: 0.3909 - accuracy: 0.8363
```





We couldn't see any improvements both on loss and accuracy, so we can conclude that the added layer is not usefull.

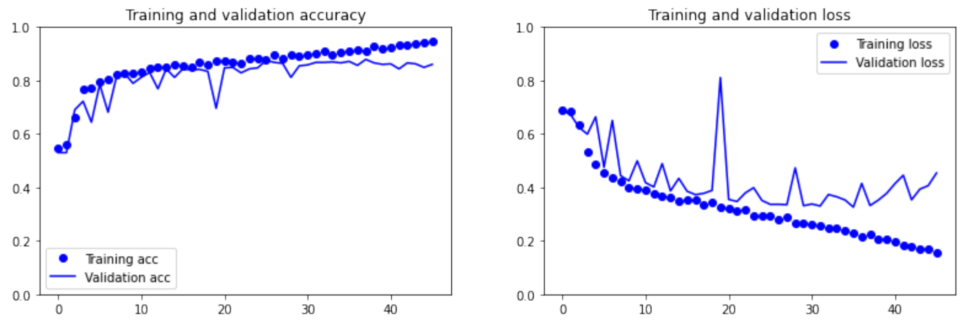
3.1.5 Model 5

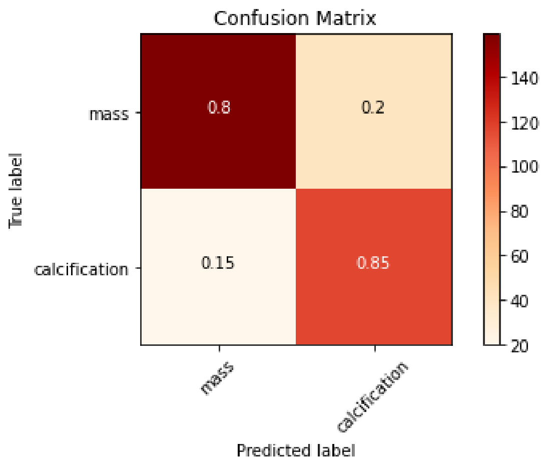
In the last trial we change the activation function of the last layer from *sigmoid* to *softmax*, but also this change brought no improvement.

- Optimizer: RMSProp, learning rate = 1e-4
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1
- Activation function: softmax

```
              precision    recall  f1-score   support
```

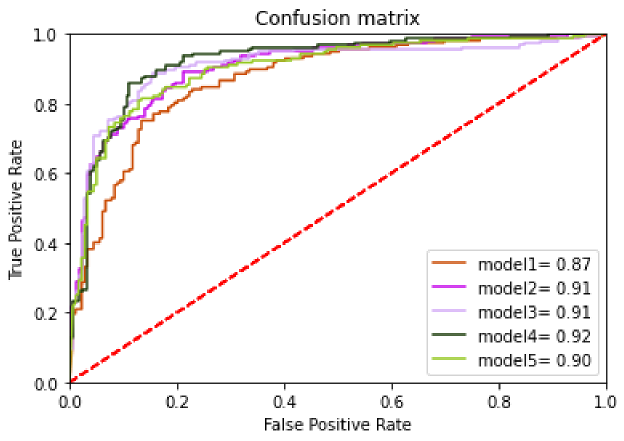
```
           0       0.81      0.89      0.85       179
           1       0.86      0.76      0.81       157

    accuracy                           0.83       336
   macro avg       0.84      0.83      0.83       336
weighted avg       0.84      0.83      0.83       336

0s 44ms/step - loss: 0.4027 - accuracy: 0.8333
```





3.2 Conclusion task 2.1

We ploted the roc curve related to each of the built models and we compared them.


Confusion matrix
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
model1= 0.87
model2= 0.91
model3= 0.91
model4= 0.92
model5= 0.90

3.3 Task 2.2, Benign - Malignant

Also in this case we set an early stopping condition, reusing the same hyperparameters.

3.3.1 Model 1

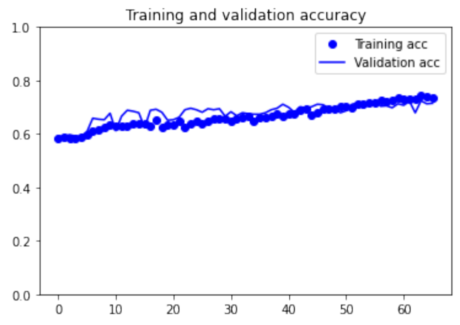
In the first trial of benign-malignant classification we built a different model, composed by less layers.

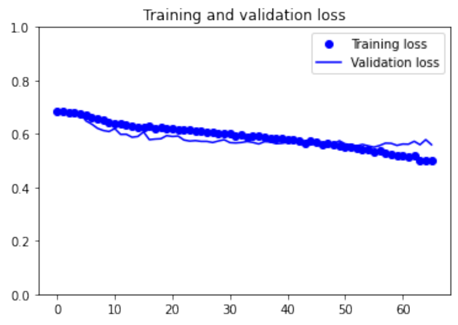
- Optimizer: RMSProp, learning rate = 1e-4
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1
- Dropout layer: rate = 0.5
- Activation function: softmax

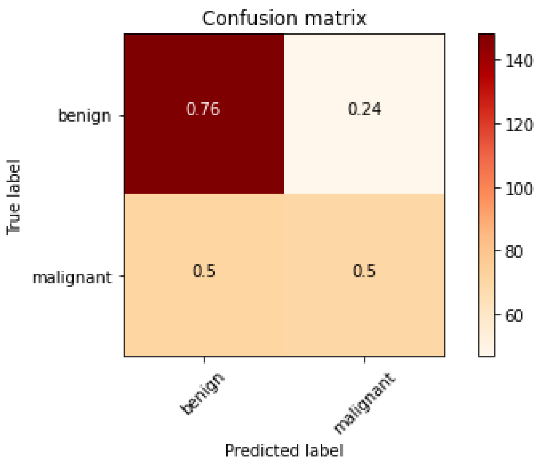
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 150, 150, 32)      320
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 37, 37, 64)        36928
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 18, 18, 64)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 18, 18, 128)       73856
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 9, 9, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 10368)             0
_________________________________________________________________
dense_8 (Dense)              (None, 256)               2654464
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 514
```

```
=================================================================
Total params: 2,784,578
Trainable params: 2,784,578
Non-trainable params: 0
_________________________________________________________________
```

```
              precision    recall  f1-score   support

           0       0.76      0.68      0.71       219
           1       0.50      0.60      0.54       117

    accuracy                           0.65       336
   macro avg       0.63      0.64      0.63       336
weighted avg       0.67      0.65      0.65       336

0s 9ms/step - loss: 0.6349 - accuracy: 0.6488
```





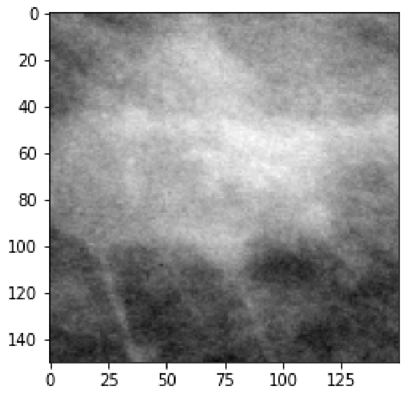


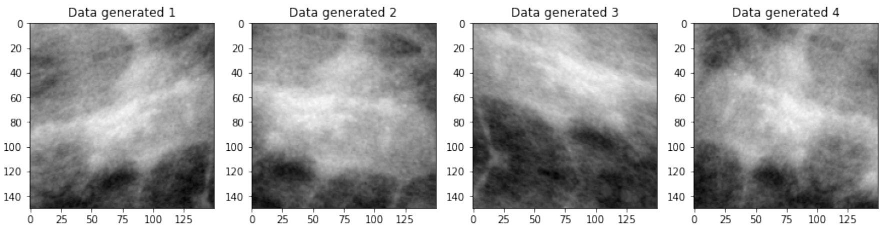
3.3.2 Deal with unbalanced dataset, using data augmentation

From the confusion matrix of the first trial we can observe the major part of the wrong classification has been computed on the class 'Malignant'. This could be a problem, considering that these cases are more important to detect.
In order to try to solve this unbalanced classification, we compute a data augmentation task, using *ImageDataGenerator* object of keras.preprocessing package.

```
dataGenerator = ImageDataGenerator(
      rotation_range=45,
      width_shift_range=0.15,
      height_shift_range=0.15,
      shear_range=20,
      zoom_range=0.15,
      horizontal_flip=True,
      fill_mode='reflect')
```

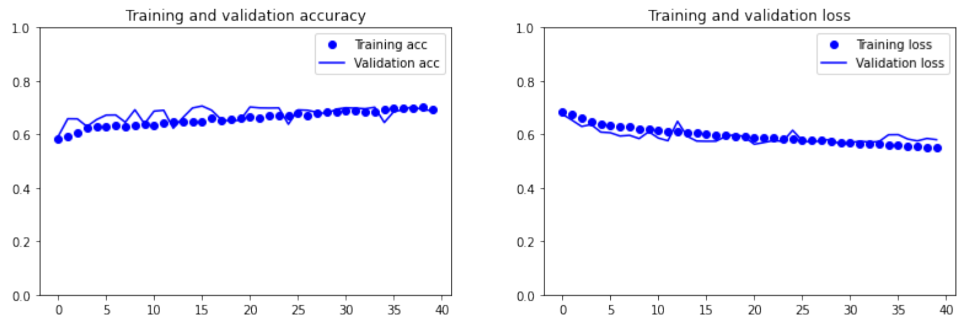
To set the parameters of ImageDataGenerator, we took inspiration from "Multi-View Feature Fusion Based Four ViewsModel for Mammogram Classification UsingConvolutional Neural Network" paper and we applied some little modifications.
The following figure shows an example of the generation: the orginal image of the dataset and the generated ones, thanks to ImageDataGenerator.

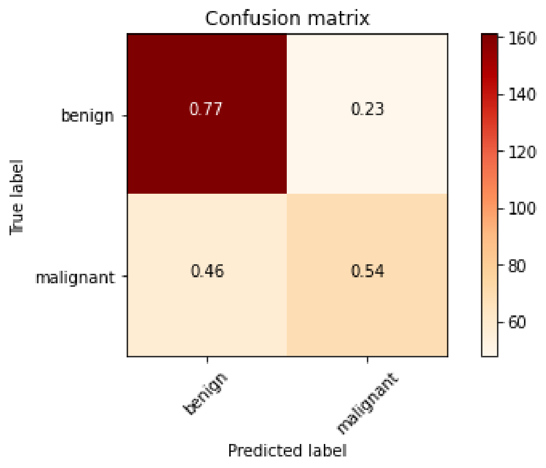




3.3.3 Model 1, trained on augmented data

```
              precision    recall  f1-score   support

           0       0.77      0.74      0.75       219
           1       0.54      0.59      0.57       117

    accuracy                           0.68       336
   macro avg       0.66      0.66      0.66       336
weighted avg       0.69      0.68      0.69       336

0s 9ms/step - loss: 0.6145 - accuracy: 0.6845
```





We improved our classifier, obtaining an increasing about 4% on the accuracy value and a decreasing on the loss value of 3%.

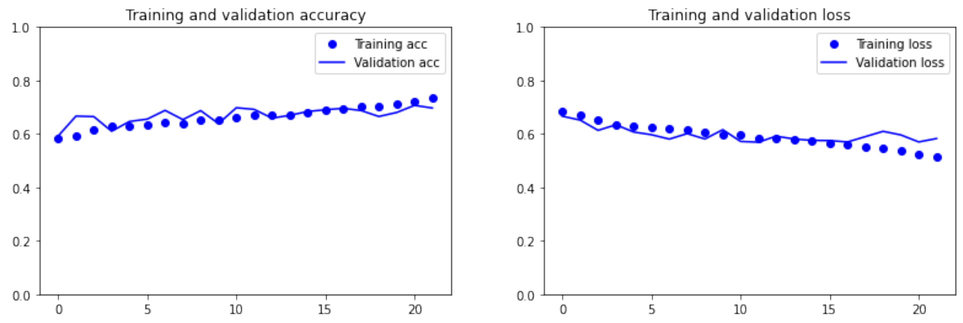
3.3.4 Model 2, trained on augmented data

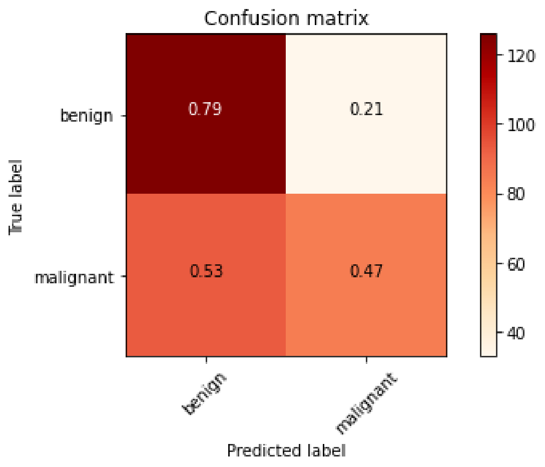
Comparing the results of the previous model trained on augmeneted data and orginal data, we decided to use the data augmentation techique to train all the following models of this task.

- Optimizer: RMSProp, learning rate = 1e-4
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 2
- Activation function: sigmoid

```
              precision    recall  f1-score   support
```

```
           0       0.79      0.58      0.67       219
           1       0.47      0.72      0.57       117

    accuracy                           0.62       336
   macro avg       0.63      0.65      0.62       336
weighted avg       0.68      0.62      0.63       336

0s 21ms/step - loss: 0.6429 - accuracy: 0.6250
```





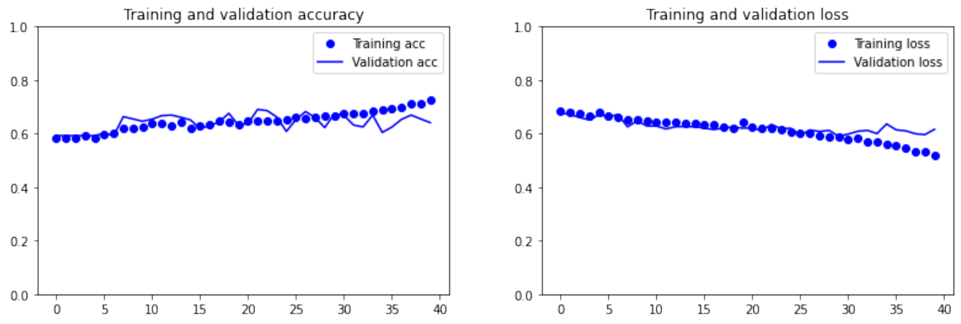
Complicating the architecture of the network, adding more layers, we didn't improve the results.

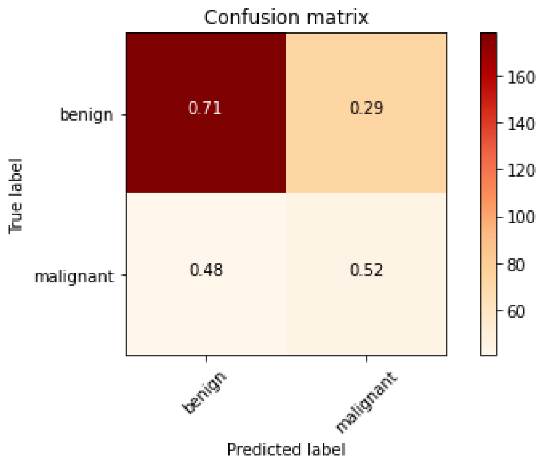
3.3.5 Model 3, trained on augmented data

- Optimizer: Adam, learning rate = 0.001 (default)
- Early stopping condition: monitoring loss on validation set, patience = 10

- Dense hidden layers: 2
- Dropout layer between the 2 dense layers: rate = 0.2
- Activation function: sigmoid

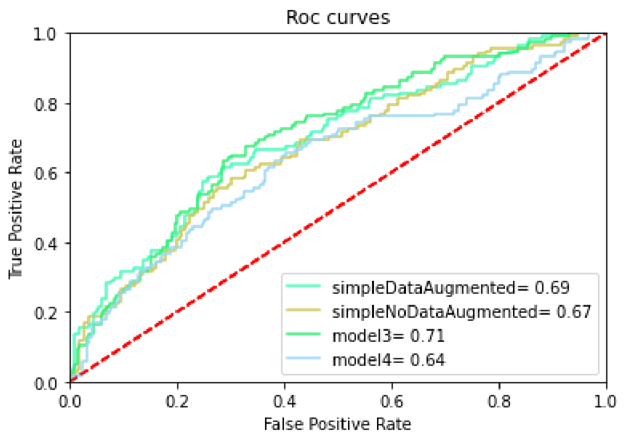
```
              precision    recall  f1-score   support

           0       0.71      0.81      0.76       219
           1       0.52      0.38      0.44       117

    accuracy                           0.66       336
   macro avg       0.61      0.59      0.60       336
weighted avg       0.64      0.66      0.65       336

0s 21ms/step - loss: 0.6186 - accuracy: 0.6607
```





3.4 Conclusion task 2.2

We ploted the roc curve related to each of the built models and we compared them.



We select the first model as best classifier, despite its area under the roc is less than the third one, because it classifies better the class Malignant.
The area under the roc equal to 0.71 related to the third classifier is given basically by the high accuracy in the correct classification of class Benign.

4 Task 3

Task 3 consists in development different models exploiting a pretrained convolotional neural networks able to compute the following classification:

- Mass - Calcification
- Benign - Malignant

Reading different papers and guides, we identified the most common used pretrained networks for images classification:

- ResNet
- GoogleNet
- VGG

For each of these networks, we used different versions and variants, for ResNet we tried ResNet50 and ResNet152; for GoogleNet we used the most recent variant InceptionV3 (while GoogleNet is called InceptionV1); for VGG we use VGG16 and VGG19.
Considering we used some pretrained networks, we followed the good practise and we decreased the learning rate respect to the task 2.1.
Another consideration to make is to note that all each pretrained neural network that we imported on the project, has been trained on a dataset composed by colored RBG images. Instead our dataset is composed by grayscale images; we found a solution which consists in replicating three times the image over the tree channels (Red - Green - Blue).

4.1 Task 3.1, Mass - Calcification

We tried to solve this task following two approaches:

- Features extraction
- Fine tuning

We trained all the models of this task on an augmented dataset, generated using the same value of the DataImageGenerator of the task 2.2.

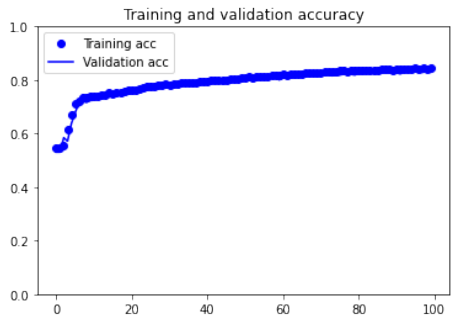
4.1.1 Model 1, VGG16 features extraction

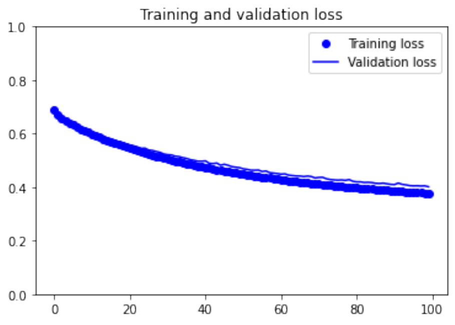
The first pretrained network used is VGG16. We cobined this pretrained network (freezing all layers) to a fully connected classifier, composed by two hidden layers.

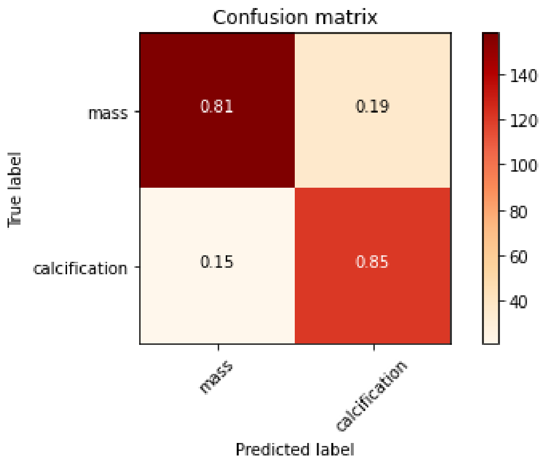
- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10

- Dense hidden layers: 2
- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.85       179
           1       0.85      0.77      0.81       157

    accuracy                           0.83       336
   macro avg       0.83      0.83      0.83       336
weighted avg       0.83      0.83      0.83       336

1s 62ms/step - loss: 0.3738 - accuracy: 0.8304
```





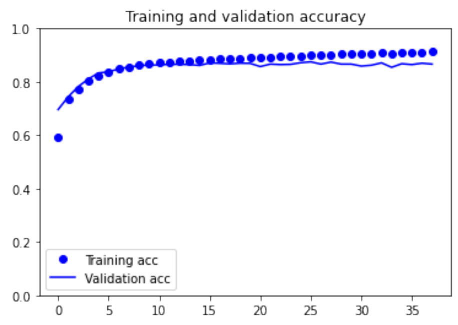


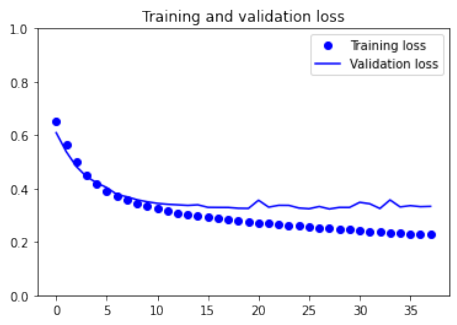
4.1.2 Model 1, VGG16 fine tuning

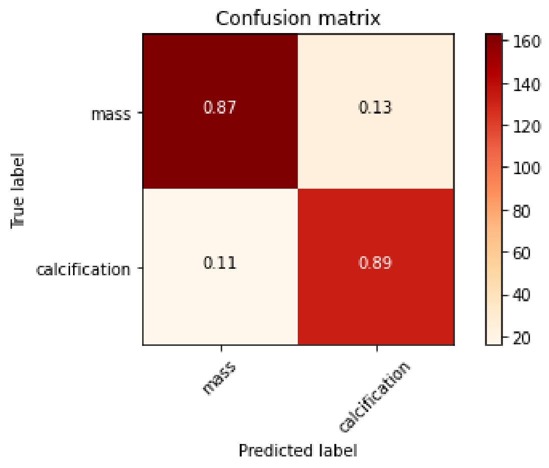
We freezed all the layers of the original model, except the first layer of the last block: *block5_conv1*.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 2
- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.87      0.91      0.89       179
           1       0.89      0.85      0.87       157

    accuracy                           0.88       336
   macro avg       0.88      0.88      0.88       336
weighted avg       0.88      0.88      0.88       336

1s 62ms/step - loss: 0.2983 - accuracy: 0.8810
```

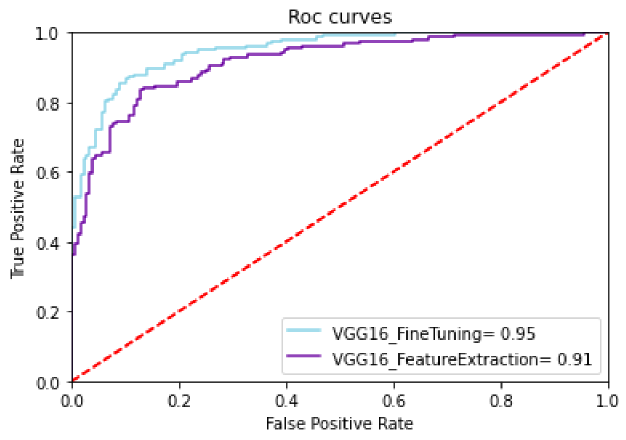






4.1.3 VGG16: features extraction VS fine tuning

Applying the fine tuning technique to VGG16, we obtained about 4% more on the area under the curve than the features extraction technique.



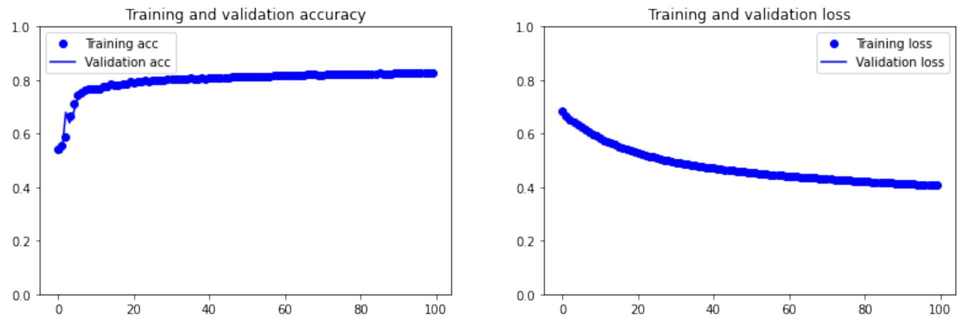
4.1.4 Model 2, VGG19 features extraction

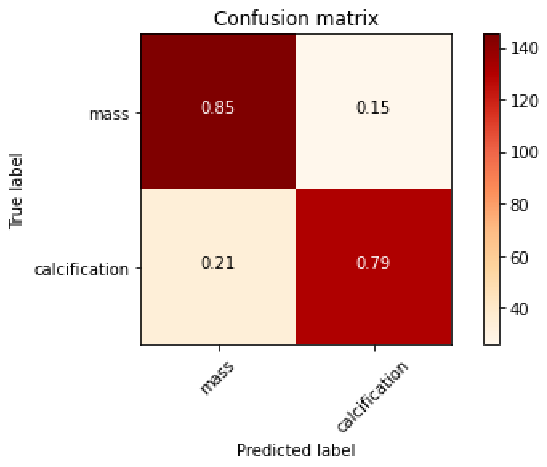
The second pretrained network used, as alternative to VGG16, is VGG19. Also in this case we cobined this pretrained network to a fully connected classifier, composed by two hidden layers, than we freezed all the layers of the original model, in order to implement the features extraction technique.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 2

- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.85      0.81      0.83       179
           1       0.79      0.83      0.81       157

    accuracy                           0.82       336
   macro avg       0.82      0.82      0.82       336
weighted avg       0.82      0.82      0.82       336

1s 74ms/step - loss: 0.4191 - accuracy: 0.8214
```



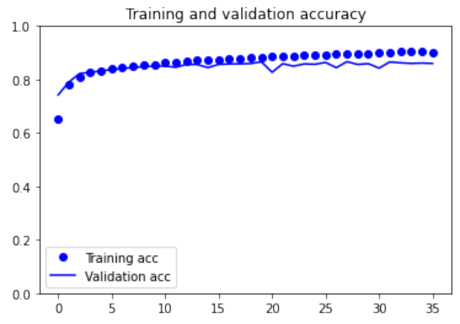


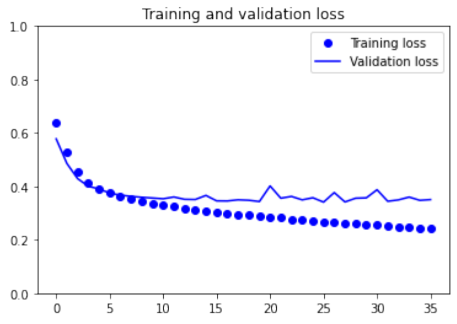
4.1.5 Model 2, VGG19 fine tuning

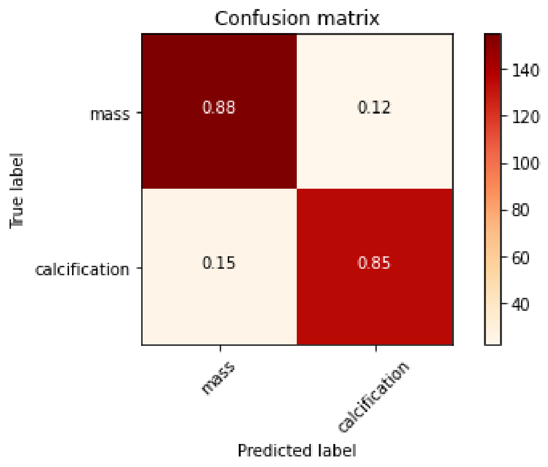
We applied the fine tuning also to this pretrained neural network: we freezed all the layers of the original model, except the first layer of the last block: *block5_conv1*.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 2
- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.88      0.87      0.87       179
           1       0.85      0.86      0.85       157

    accuracy                           0.86       336
   macro avg       0.86      0.86      0.86       336
weighted avg       0.86      0.86      0.86       336

1s 75ms/step - loss: 0.3346 - accuracy: 0.8631
```

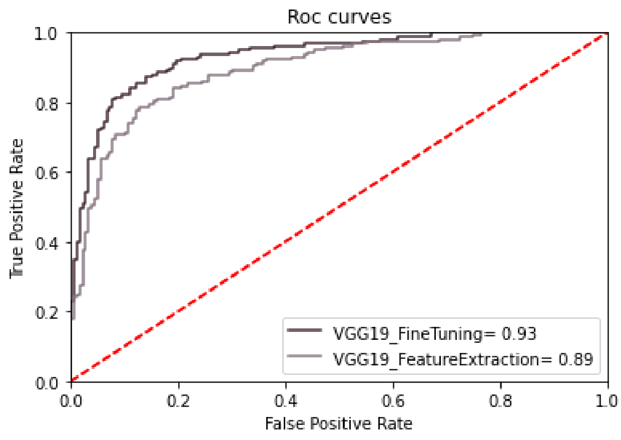






4.1.6 VGG19: features extraction VS fine tuning

Applying the fine tuning technique to VGG19, we obtained about 4% more on the area under the curve than the features extraction technique.



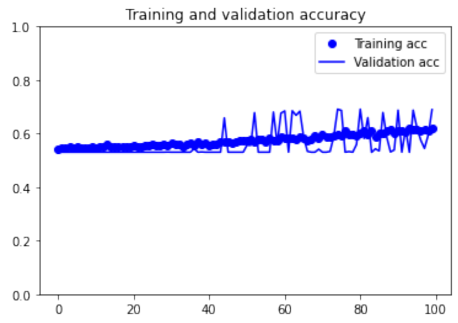
4.1.7 Model 3, ResNet50 features extraction

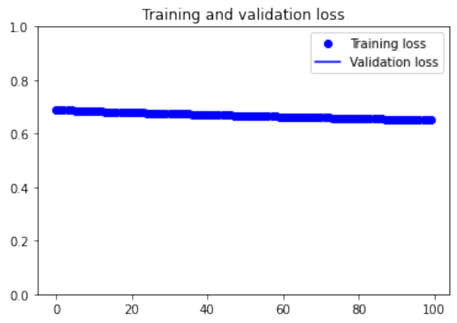
The third pretrained network used is ResNet50. In this case we cobined this pretrained network to a fully connected classifier, composed by a single hidden layers, than we freezed all the layers of the original model, in order to implement the features extraction technique.

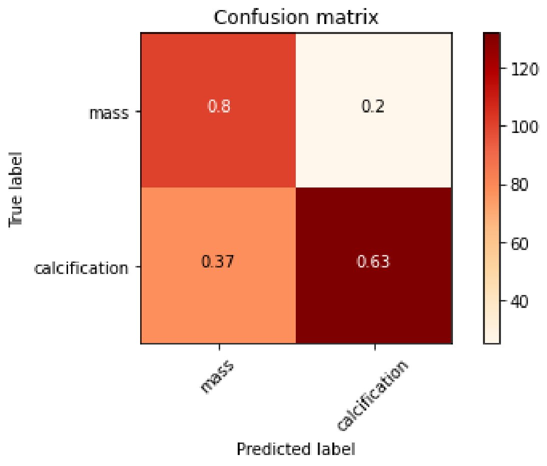
- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1

- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.80      0.56      0.66       179
           1       0.63      0.84      0.72       157

    accuracy                           0.69       336
   macro avg       0.72      0.70      0.69       336
weighted avg       0.72      0.69      0.69       336

1s 50ms/step - loss: 0.6555 - accuracy: 0.6935
```





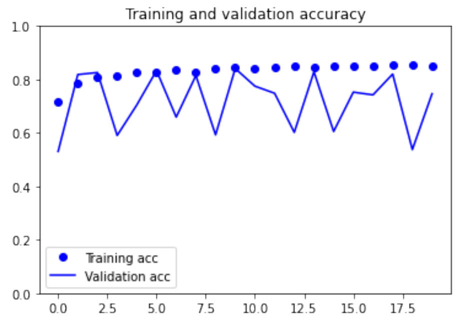


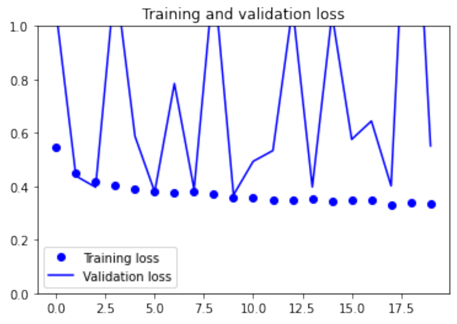
4.1.8 Model 3, ResNet50 fine tuning

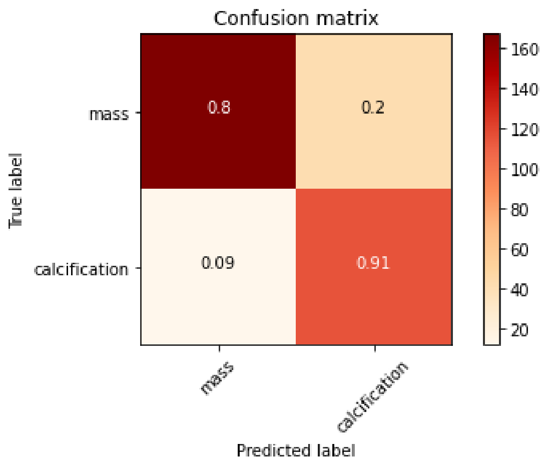
We applied the fine tuning also to this pretrained neural network: we freezed all the layers of the original model, except the first layer of the last block: *block5_conv1*.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1
- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.80      0.93      0.86       179
           1       0.91      0.73      0.81       157

    accuracy                           0.84       336
   macro avg       0.85      0.83      0.84       336
weighted avg       0.85      0.84      0.84       336

1s 51ms/step - loss: 0.3897 - accuracy: 0.8393
```

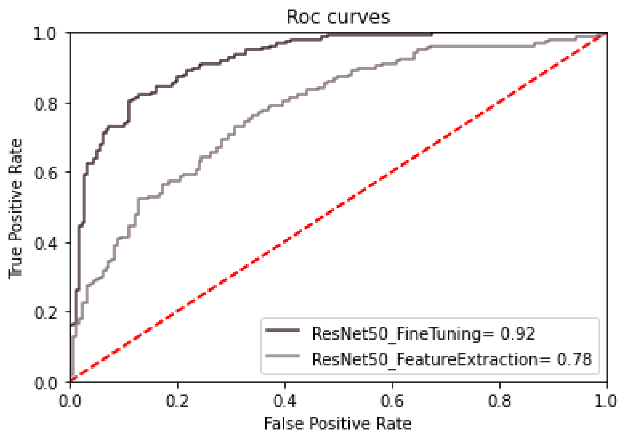






4.1.9 ResNet50: features extraction VS fine tuning

Applying the fine tuning technique to ResNet50, we obtained about 14% more on the area under the curve than the features extraction technique.



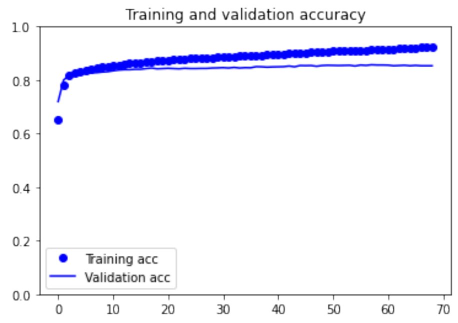
4.1.10 Model 4, InceptionV3 features extraction

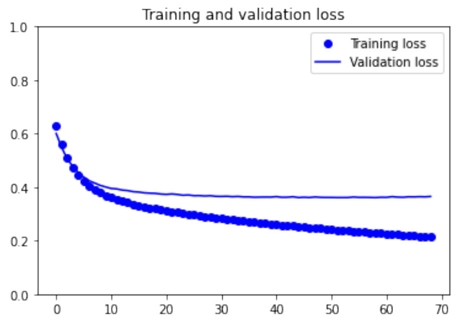
The fourth and last pretrained network used is InceptionV3. Also this case we cobined this pretrained network to a fully connected classifier, composed by a single hidden layers, as in the previous case. Than we freezed all the layers of the original model, in order to implement the features extraction technique.

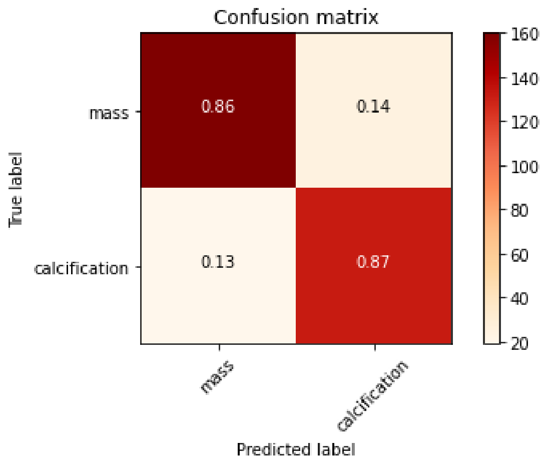
- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1

- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.86      0.89      0.88       179
           1       0.87      0.84      0.86       157

    accuracy                           0.87       336
   macro avg       0.87      0.87      0.87       336
weighted avg       0.87      0.87      0.87       336

0s 31ms/step - loss: 0.3268 - accuracy: 0.8690
```





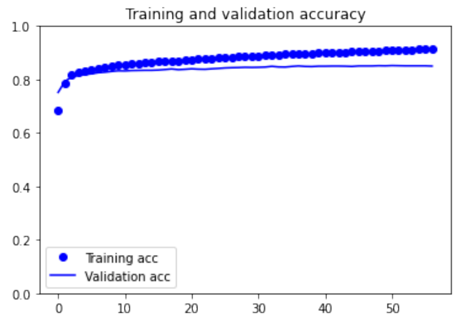


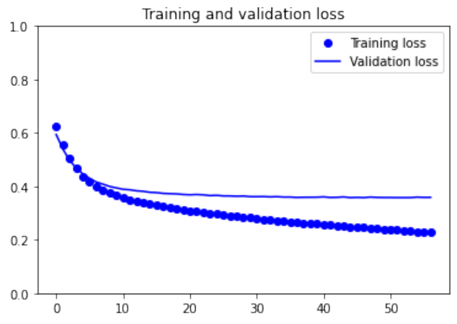
4.1.11 Model 4, InceptionV3 fine tuning

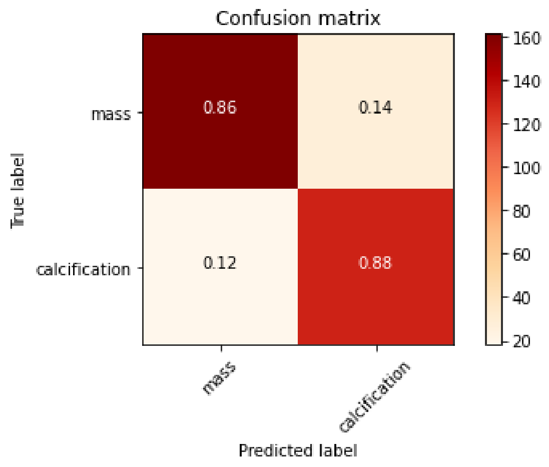
We applied the fine tuning also to this pretrained neural network: we freezed all the layers of the original model, except the first layer of the last block: *conv2d_93*.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1
- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       179
           1       0.88      0.83      0.85       157

    accuracy                           0.87       336
   macro avg       0.87      0.86      0.86       336
weighted avg       0.87      0.87      0.87       336

0s 31ms/step - loss: 0.3288 - accuracy: 0.8661
```

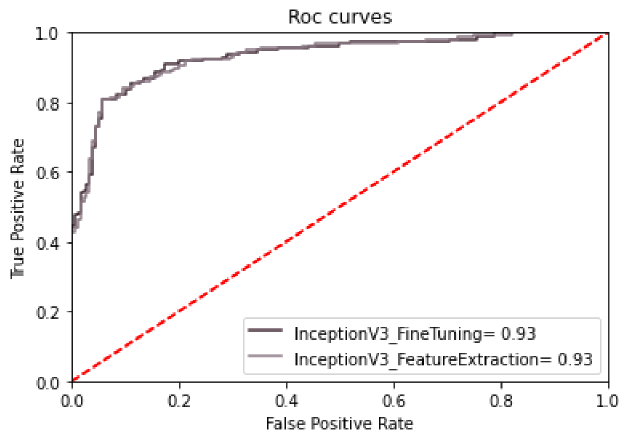






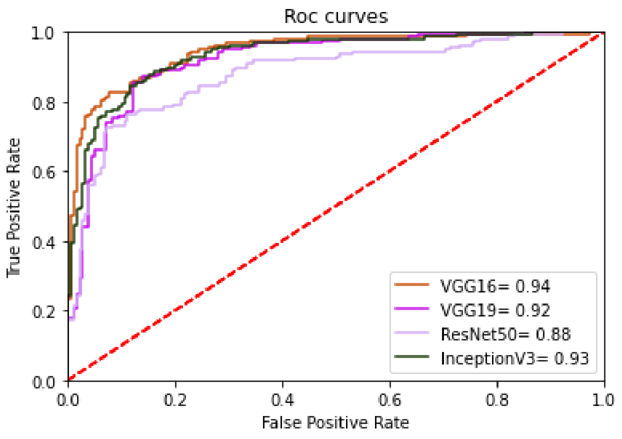
4.1.12 InceptionV3: features extraction VS fine tuning

Applying the features extraction and fine tuning techniques to InceptionV3 neural network, we obtained the same results in terms of accurancy and loss.



4.2 Comparison between the models of the task 3.1

We ploted the roc curve related to each of the models built using the fine tuning technique, because it gives better models than features extraction (except for InceptionV3) and we compared them.



We saved the models based on VGG16, VGG19 and InceptionV3, in order to use them in the task 5.

4.3 Task 3.2, Benign - Malignant

Also in this case we tried to solve this classification problem with both features extraction and fine tuning, but unlike the previous task, in some cases we obtained better results using features extraction technique, others using fine tuning.
Moreover we imported the same pretrained networks, trained on imagenet dataset. We combined two approaches, in order to rebalance the two classes.

- Data augmentation, using *ImageDataGenerator*, basically to increase the dimension of the dataset, in order to use the fine tuning technique.
- Assign different weights to the two classes, basically to give more importance to the malignant one, that is more critical to detect.

We tried some formulas, in order to try different values of weights :

```
weightsBenign = malignantNum / (malignantNum + benignNum)
weightsMalignant = benignNum / (malignantNum + benignNum)
weights = { 0: weightsBenign * 2,  1: (weightsMalignant + 0.3) * 2}

0.828101644245142
1.7718983557548578


weightsBenign = malignantNum / (malignantNum + benignNum)
weightsMalignant = benignNum / (malignantNum + benignNum)
weights = { 0: weightsBenign - .05, 1: weightsMalignant + .05}

0.364050822122571
0.635949177877429


weightsBenign = malignantNum / min(malignantNum, benignNum)
weightsMalignant = benignNum / min(malignantNum, benignNum)
weights = { 0: weightsBenign, 1: weightsMalignant}

1.0
1.4151624548736461
```

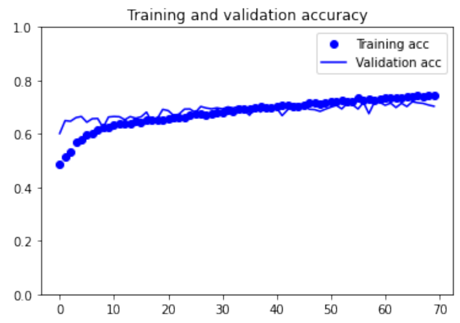
After some experiments we decided to use the third couple of weights:
weights = { 0: 1.0 , 1: 1.415... }

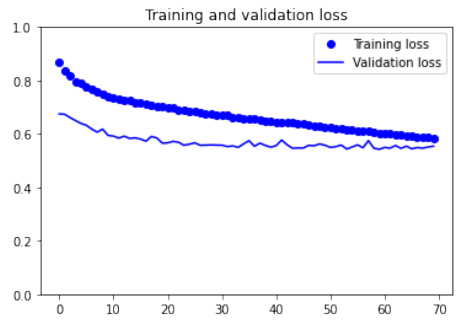
4.3.1 Model 1, VGG16 fine tuning

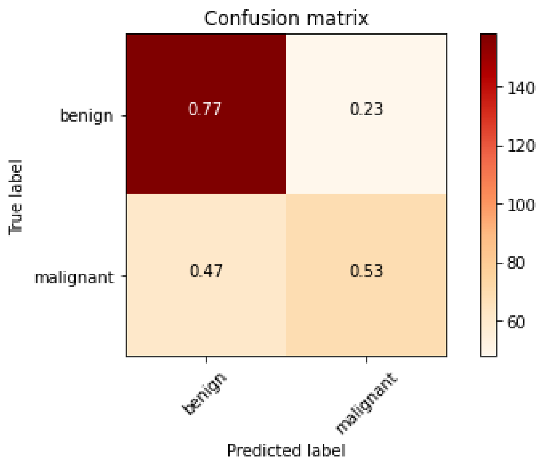
We freezed all the layers of the original model, except the first layer of the last block: *block5_conv1*.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 2
- Dropout layers between the dense hidden one: rate = 0.5
- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.77      0.72      0.74       219
           1       0.53      0.59      0.56       117

    accuracy                           0.68       336
   macro avg       0.65      0.66      0.65       336
weighted avg       0.68      0.68      0.68       336

1s 63ms/step - loss: 0.6062 - accuracy: 0.6756
```





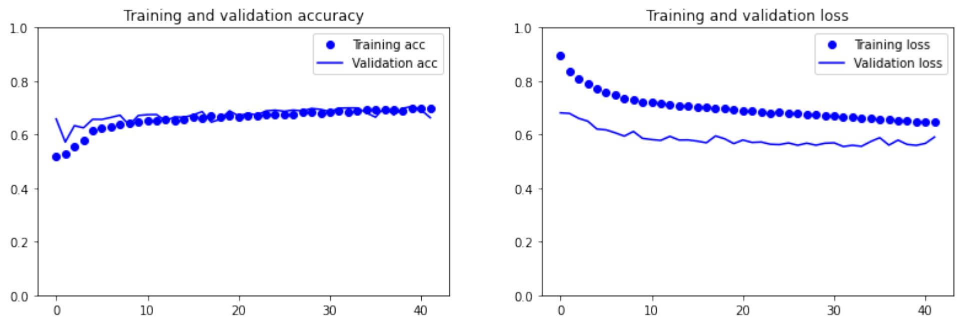


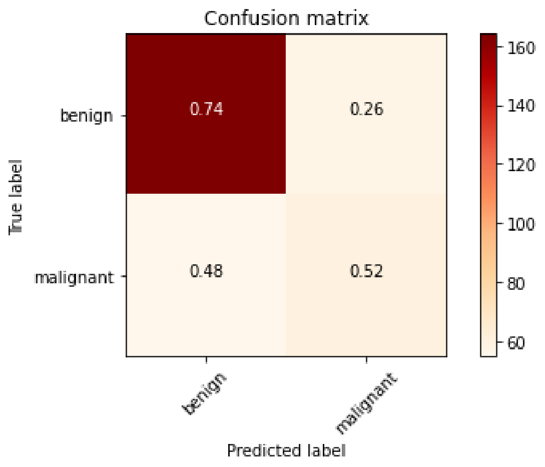
4.3.2 Model 2, VGG19 fine tuning

We applied the fine tuning also to this pretrained neural network: we freezed all the layers of the original model, except the first layer of the last block: *block5_conv1*.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 2
- Dropout layers between the dense hidden one: rate = 0.5
- Activation function: sigmoid

```
              precision    recall  f1-score   support

           0       0.74      0.75      0.75       219
           1       0.52      0.51      0.52       117

    accuracy                           0.67       336
   macro avg       0.63      0.63      0.63       336
weighted avg       0.67      0.67      0.67       336

1s 78ms/step - loss: 0.6144 - accuracy: 0.6667
```





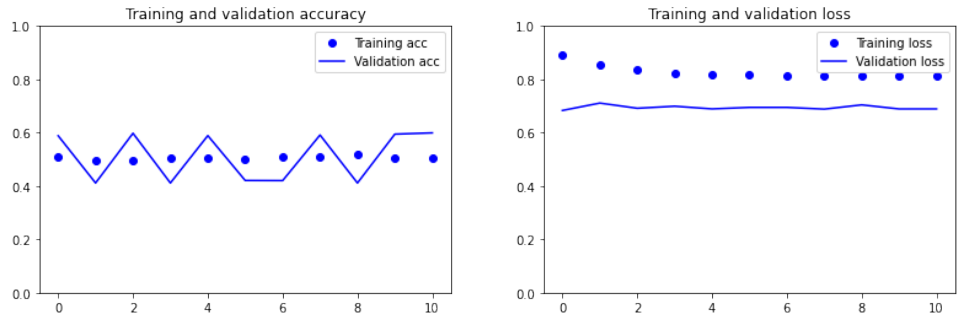
4.3.3 Model 3, ResNet50 features extraction

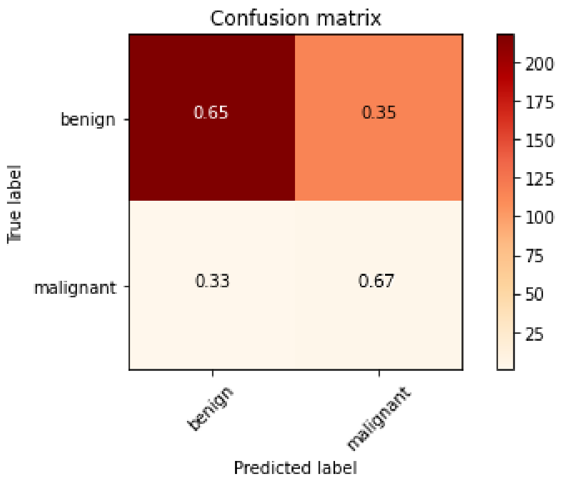
We applied the fine tuning also to this pretrained neural network: we freezed all the layers of the original model.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10
- Dense hidden layers: 1
- Dropout layers between the dense hidden layer and the output one: rate = 0.5
- Activation function: sigmoid

```
              precision    recall  f1-score   support
```

```
           0       0.65      1.00      0.79       219
           1       0.67      0.02      0.03       117

    accuracy                           0.65       336
   macro avg       0.66      0.51      0.41       336
weighted avg       0.66      0.65      0.53       336

1s 50ms/step - loss: 0.6869 - accuracy: 0.6548
```





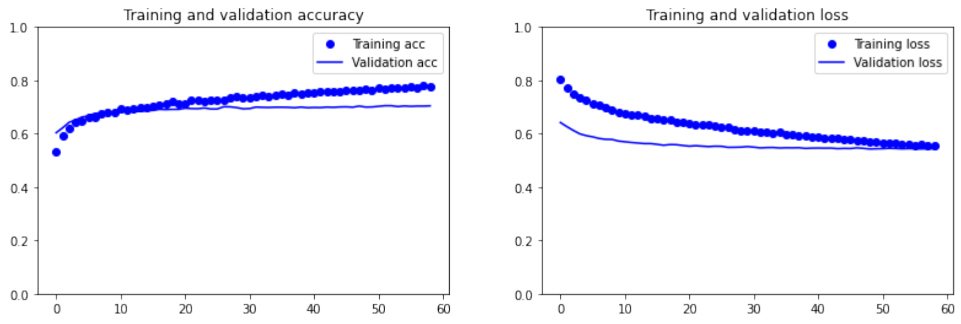
4.3.4 Model 4, InceptionV3 features extraction

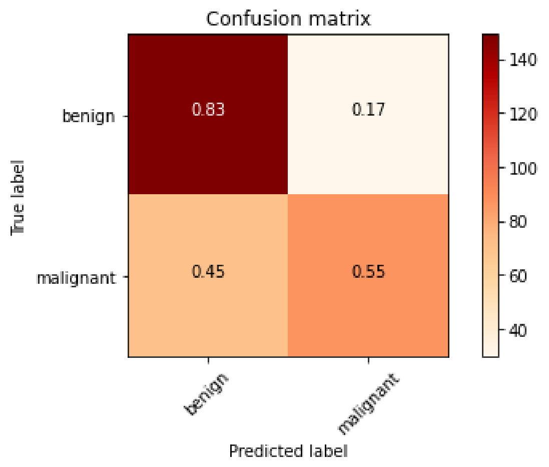
We applied the fine tuning also to this pretrained neural network: we freezed all the layers of the original model.

- Optimizer: RMSProp, learning rate = 2e-6
- Early stopping condition: monitoring loss on validation set, patience = 10

- Dense hidden layers: 1
- Dropout layers between the dense hidden layer and the output one: rate = 0.5
- Activation function: sigmoid

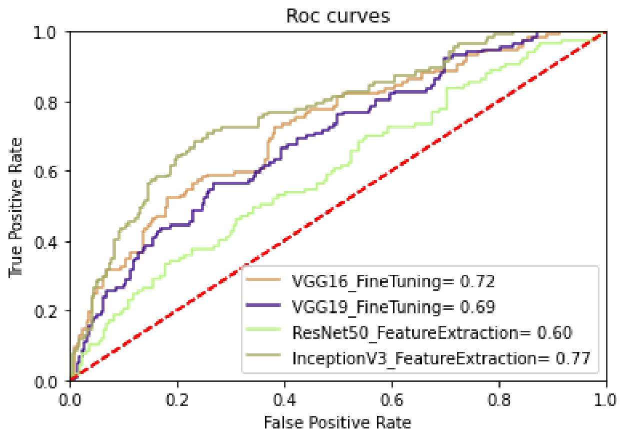
```
              precision    recall  f1-score   support

           0       0.83      0.68      0.75       219
           1       0.55      0.74      0.64       117

    accuracy                           0.70       336
   macro avg       0.69      0.71      0.69       336
weighted avg       0.74      0.70      0.71       336

0s 30ms/step - loss: 0.5773 - accuracy: 0.7024
```





4.4 Comparison between the models of the task 3.2

We ploted the roc curve related to each of the models shown in the previous paragraphs and we compared them.



We saved the top 3 models: those based on VGG16 (fine tuning), VGG19 (fine tuning), and InceptionV3 (features extractions), in order to use them in the task 5.

5 Task 4

We tried to increase the results of task 3.2 exploiting the baseline patches, using different approaches:

- **Siamese network**
- **Hilghlight the abnormality portion** using the baseline patches and a pretrained network

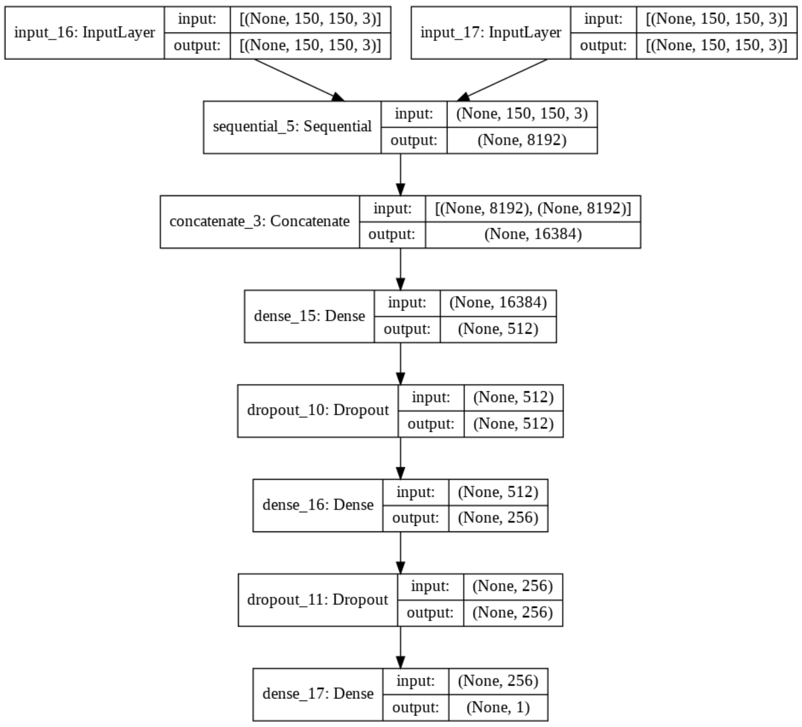
5.1 Siamese architecture

We implemented a siamese neural network, as shown on the following picture; where the two sisters networks are VGG16 pretrained on the dataset imagenet. We tried to impelment the merge layer following two different approaches:

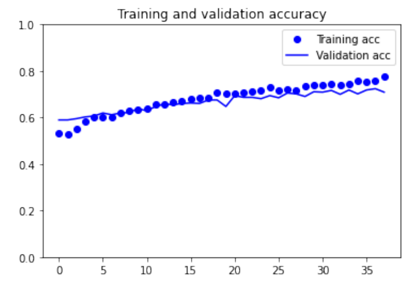
- Concatenate the output of the siters networks
- Compute the difference element by element between the output of the siters networks

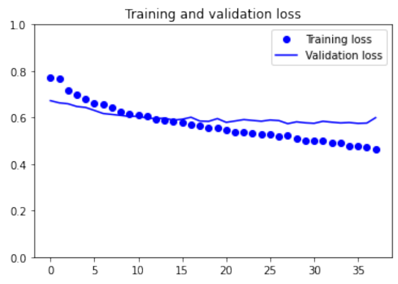
Finally there is the classifier composed by a single dense hidden layer.

5.1.1 Siamese, implementing the concatenation

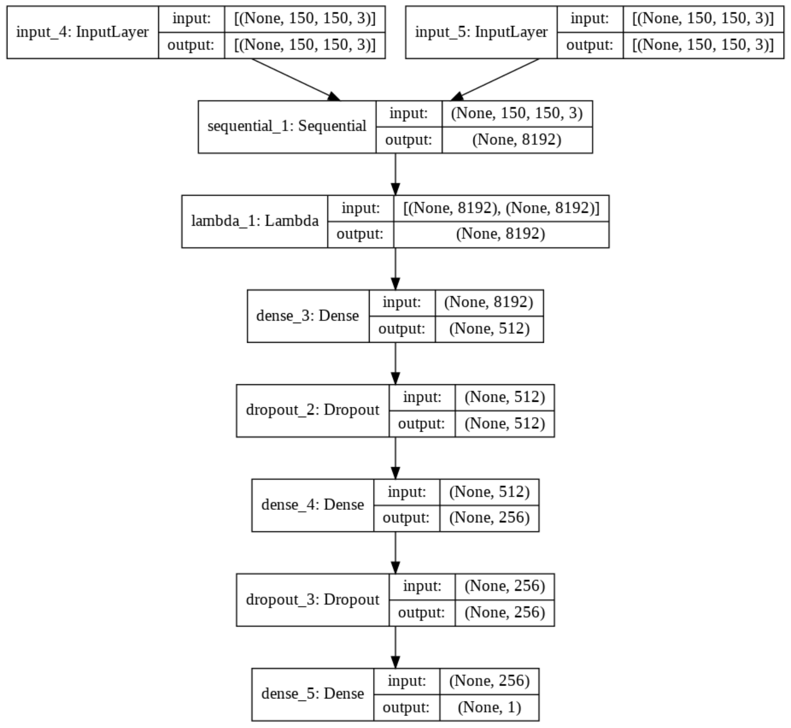


```
              precision    recall  f1-score   support
           0       0.78      0.73      0.75       219
           1       0.55      0.62      0.58       117

    accuracy                           0.69       336
   macro avg       0.67      0.67      0.67       336
weighted avg       0.70      0.69      0.69       336

1s 133ms/step - loss: 0.5700 - accuracy: 0.6905
```

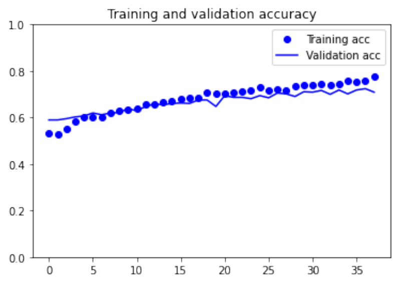


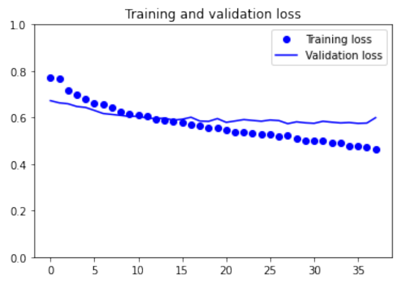


5.1.2 Siamese, implementing the difference


input_4: InputLayer | input: [(None, 150, 150, 3)] | output: [(None, 150, 150, 3)]
input_5: InputLayer | input: [(None, 150, 150, 3)] | output: [(None, 150, 150, 3)]
sequential_1: Sequential | input: (None, 150, 150, 3) | output: (None, 8192)
lambda_1: Lambda | input: [(None, 8192), (None, 8192)] | output: (None, 8192)
dense_3: Dense | input: (None, 8192) | output: (None, 512)
dropout_2: Dropout | input: (None, 512) | output: (None, 512)
dense_4: Dense | input: (None, 512) | output: (None, 256)
dropout_3: Dropout | input: (None, 256) | output: (None, 256)
dense_5: Dense | input: (None, 256) | output: (None, 1)

```
precision    recall  f1-score   support
           0       0.73      0.70      0.71       219
           1       0.48      0.52      0.50       117

    accuracy                           0.64       336
   macro avg       0.61      0.61      0.61       336
weighted avg       0.64      0.64      0.64       336

1s 133ms/step - loss: 0.6231 - accuracy: 0.6369
```





5.1.3 Conclusion of Siamese network

After implementing this architecture, we compare the results obtained with those obtained on task 3.2 and we increased the accurancy about 2 - 3% respect to VGG16, VGG19 and ResNet50 and the same results respect to the InceptionV3.

5.2 Pretrained network

The other approach we followed to use the baseline patches to improve the accuracy of our models, trying to remove the healthy tissue part from the abnormality images.
The intuition of this technique is the following: we aspected an uniform distribution of color in the baseline images and some small portions of healthy tissue in the abnormality ones, so we would highlight the abnormality, computing the difference berween the two images.

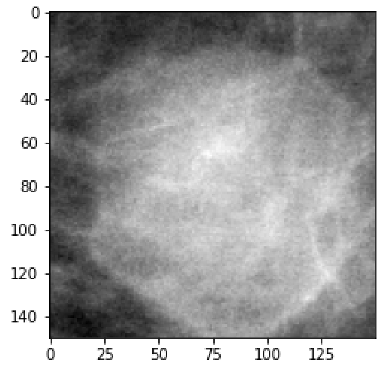


Figure 1: Abnormality

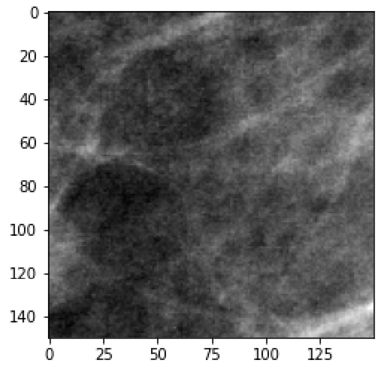


Figure 2: Baseline patch

By observing the baseline image, we can say that the color is not plane as we expected, so we compute the mean value of the matrix, in order to force color uniformity.

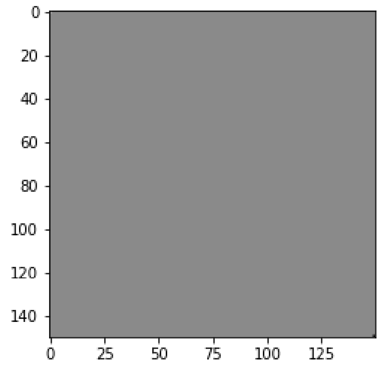


Figure 3: Mean value of the healthy image

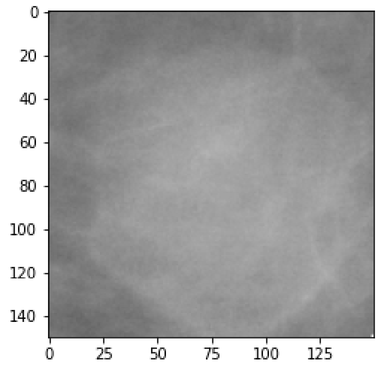


Figure 4: Difference between abnormality and the mean of baseline

After implementing this intuition we obtained the same value of the accuracy and an increase of the loss. Trying to emphasize the abnormality, we actually remove part of infomation.

6 Task 5, Ensemble

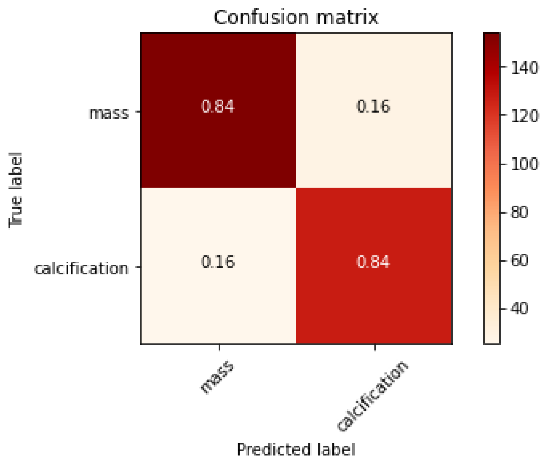
In this task we combined different models that we saved for each classification problem, except for task 2.2, where we saved a single model, implementing the following approaches:

- Average voting
- Majoring voting

6.1 Enseble, task 2.1

6.1.1 Average Voting

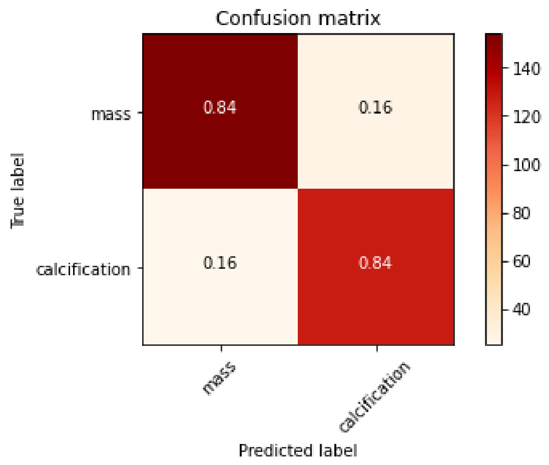
```
              precision    recall  f1-score   support

           0       0.83      0.87      0.85       179
           1       0.84      0.80      0.82       157

    accuracy                           0.84       336
   macro avg       0.84      0.83      0.84       336
weighted avg       0.84      0.84      0.84       336
```



6.1.2 Majority Voting

```
              precision    recall  f1-score   support

           0       0.84      0.86      0.85       179
           1       0.84      0.82      0.83       157
```

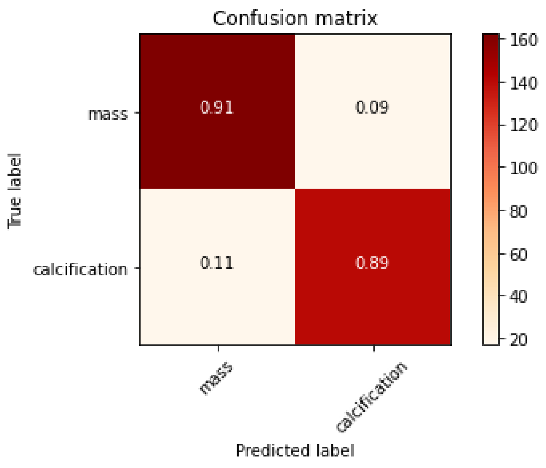
```
    accuracy                           0.84       336
   macro avg       0.84      0.84      0.84       336
weighted avg       0.84      0.84      0.84       336
```



6.2 Enseble, task 3.1

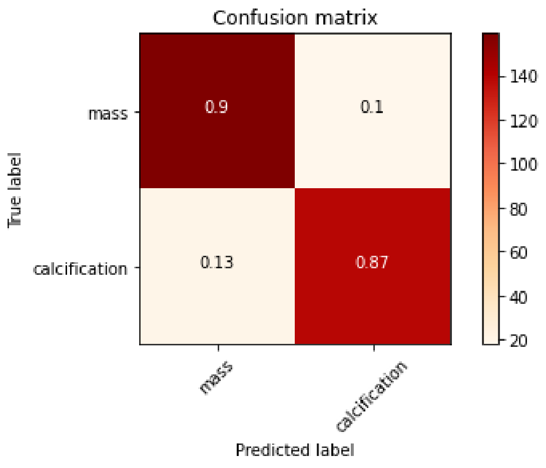
6.2.1 Average Voting

```
              precision    recall  f1-score   support

           0       0.91      0.91      0.91       179
           1       0.89      0.89      0.89       157

    accuracy                           0.90       336
   macro avg       0.90      0.90      0.90       336
weighted avg       0.90      0.90      0.90       336
```



6.2.2 Majority Voting

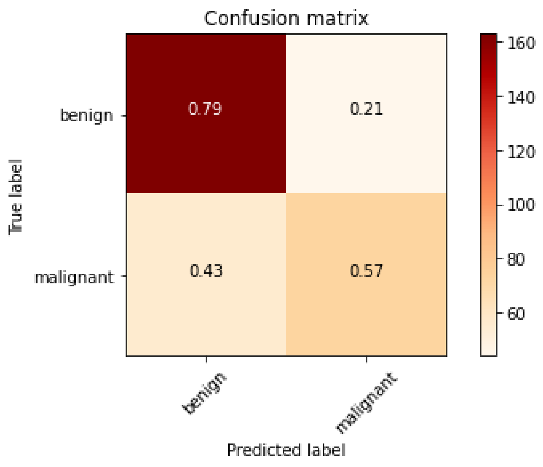
```
              precision    recall  f1-score   support

           0       0.90      0.89      0.89       179
           1       0.87      0.89      0.88       157

    accuracy                           0.89       336
   macro avg       0.89      0.89      0.89       336
weighted avg       0.89      0.89      0.89       336
```



6.3 Enseble, task 3.2

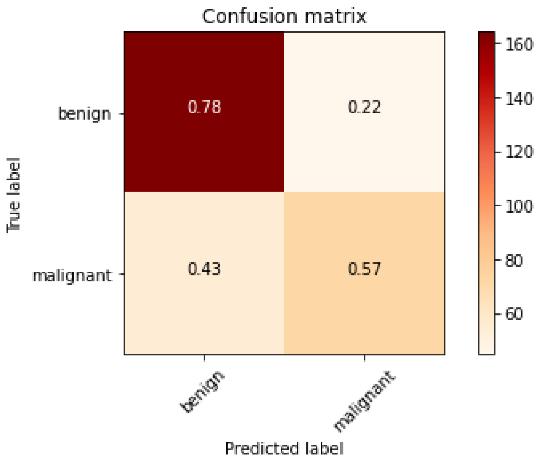
6.3.1 Average Voting

```
              precision    recall  f1-score   support

           0       0.79      0.74      0.77       219
           1       0.57      0.62      0.59       117

    accuracy                           0.70       336
   macro avg       0.68      0.68      0.68       336
weighted avg       0.71      0.70      0.71       336
```



6.3.2 Majority Voting

```
              precision    recall  f1-score   support

           0       0.78      0.75      0.77       219
           1       0.57      0.62      0.59       117

    accuracy                           0.70       336
   macro avg       0.68      0.68      0.68       336
weighted avg       0.71      0.70      0.71       336
```



6.4 Conclusion Ensembling

Looking the results obtained in this task, we can observe in same cases a small increase on the accuracy respect of some single models, in others the performaces remains unchanged.
Generally the average voting gives better results than majority voting, about 1%.