FINAL PROJECT
COMPUTATIONAL INTELLIGENCE

UNIVERSITY OF PISA

DEPARTMENT OF INFORMATION ENGINEERING

CNN for Medical Imaging Analysis

Authors:
Leonardo Lai

Email:
leonardo.lai@live.com

Date: January 16, 2020

Contents

1 Introduction **4**

2 State of the art **4**

3 Data pre-processing **5**

4 CNN from scratch **7**
4.1 2 class 7
4.1.1 Experiment 0 - Simple model 7
4.1.2 Experiment 1 - Dropout 9
4.1.3 Experiment 2 - Data augmentation 9
4.1.4 Experiment 3 - Larger network, Early stopping 11
4.1.5 Experiment 4 - Larger FC 13
4.1.6 Experiment 5 - Learning rate decay 14
4.1.7 Experiment 6 - Deeper 15
4.1.8 Experiment 7 - Larger kernel 16
4.1.9 Experiment 8 - Adam 18
4.1.10 Experiment 9 - L2 regularization 18
4.1.11 Experiment 10 - Batch normalization 19
4.1.12 Experiment 11 - BN (smaller momentum) 20
4.2 Learning rate analysis 21
4.2.1 Estimation 21
4.2.2 Verification 23
4.3 Error analysis 25
4.4 4 class 27
4.4.1 Experiment 1 - Previous best 27
4.4.2 Experiment 2 - More filters 28
4.4.3 Experiment 3 - Deeper 29

5 Pre-trained models **31**
5.1 VGG16 2-class 31
5.1.1 Feature Extraction 31
5.1.2 Fine tuning 34
5.2 VGG16 4-class 36
5.2.1 Feature Extraction 36
5.2.2 Fine tuning 37
5.3 ResNet50 2-class 39
5.3.1 Feature extraction 39
5.4 MobileNet 41
5.4.1 Feature extraction 41

6 Composite **44**

7 Baseline **45**
7.1 Dual channel CNN 45
7.2 Siamese CNN 47

8 Ensemble **49**
8.1 2-class 49
8.1.1 3-ensemble 49
8.1.2 4-ensemble 49
8.2 4-class 49
8.2.1 3-ensemble 50
8.2.2 4-ensemble 50
8.2.3 Weighted ensemble with grid search 51

1 Introduction

This project aims to perform abnormality classification in mammography by means of Convolutional Neural Networks. The dataset of interest is the *CBIS DDSM: Curated Breast Imaging Subset of Digital Database for Screening Mammography*, a collection of mammography images by Lee et al. [9]. It is an updated version of the original DDSM dataset, where all the images have been segmented and labeled, and additional information is provided in csv files. Two types of breast abnormalities are present in the pictures: mass and calcification. Both can be either benign or malignant, and the overall classification task consists in distinguishing between the four classes:

- Benign mass
- Malignant mass
- Benign calcification
- Malignant calcification

A subtask is to tell apart masses from calcifications, regardless of their malignancy.

2 State of the art

The *DDSM* is the largest mammography public dataset available at the moment, featuring 2620 cases associated to different patients. *BancoWeb* is a public database with more than 1400 breast cancer images and clinical histories, that also allows volunteers to contribute with new samples. [11]. *INBreast* contains 410 images of mass, calcification, asymmetry and normal tissues from 115 medical cases, collected in Portugal and carefully labeled by specialists [14]. Another popular but older collection is the *MIAS*, with more than 300 left and right breast images [25] from the UK. Private datasets exist as well, like the OPTIMAM by Northwestern Medicine that was recently used by Google for their work [12].
Many scholars and researchers have investigated these data over the last few years, with the aim of developing new CAD (Computer Assisted Diagnosis) techniques for breast cancer diagnosis. Several studies, especially the most recent ones, exploit deep learning models to classify the type of the abnormalities. However, approaches based on signal processing and computer vision techniques are common too.

Back in 2011, Sharkas et al. proposed a combination of DWT (Discrete Wavelet Transform), PCA (Principal Components Analysis) and SVM (Support Vector Machine) to classify normal and tumor breast tissues [19].

Another original proposal came in 2015 from Dhungel et al., who cascaded a deep convolutional network with random forests and connected component analysis (CCA), achieving state-of-the-art results on DDSM and INbreast [4].

Lèvy et al. [10] surpassed human performance in the classification of DDSM images by means of CNN, exploiting transfer learning on pretrained models like AlexNet,

the network that won ILSVRC (ImageNet Large Scale Visual Recognition Challenge) in 2012, and GoogLeNet, winner of the same competition in 2014. [8][26].

In 2018, Xi et al. used VGGNet, the winner of ImageNet challenge in 2014, to achieve a 92.53% classification accuracy [27][21]. The same authors exploited ResNet to localize the abnormalities within the full mammography images [5].

Recently, Ragab et al. extracted ROIs from mammography both manually and with threshold-based techniques, then classified them using AlexNet chained with SVM [17]. On the CBIS DDSM dataset, they claim an accuracy of 87.2% with a 0.94 AUC.

Shen et al. further extended these studies by comparing the results achieved by several state-of-the-art architectures; when averaging the top four models, they were able to obtain a 0.91 AUC [20].

In 2020, an article by Google researchers was published in Nature. The authors trained an ensemble of three models on more than 28000 mammogram images, then compared its predictions with the decisions of radiologists. The actual labels were determined by follow-up exams or biopsy. It turns out that AI beats humans in terms of sensitivity and specificity [12].

Some scholars focused addressed the scarcity of images in the DDSM dataset by proposing data augmentation techniques. Hussain et al. compared different transformations, proving that using augmentation functions which preserve a high amount of information (i.e. not too disruptive) help increasing the classification accuracy. Similar results were obtained by Costa et al. in a less extensive study [3].

3 Data pre-processing

The dataset is provided as a set of numpy arrays, containing the images and labels to use for training and testing. Before these data can be actually used as input for NN models, a few pre-processing steps are necessary. Depending on the specific classification task (e.g. mass/calcification, benign/malignant, ...), the actions to perform can be slightly different.
The following list describes the whole sequence to prepare the data:

1. Import the training and testing data as numpy arrays from shared .npy files.

2. When the baseline patches are not needed, remove them and the corresponding labels from the arrays (even indices).

3. Remap the labels depending on how many and which classes are involved in the specific classification. If the task is to just distinguish between masses and calcification, then only two labels (0-1) are needed. Conversely, four labels (0-3) are required when it is also important to discriminate benign abnormalities from malignant ones.

4. Normalize the pixel values to be in a range compatible with the chosen model. Scratch CNN models use input in the range (0,1) floating point, while VGGNet and other pretrained models are designed to work with images in (0,255)

that are further pre-processed with custom transformations (channel swapping, mean subtraction, ...).

5. Shuffle the training set and corresponding labels accordingly.
6. Split the training data into 'training' and 'validation' subsets. The former will be used to compute the loss function exploited by the optimizer, whereas the validation set is only used to monitor the actual performance on an independent set during training.
7. Instantiate Keras generators as data sources for the network. Data augmentation settings can be specified at this stage.

At the end of the pipeline, one or more of the resulting samples are effectively visualized to verify that:

1. The data is formatted as expected (size, range, ...)
2. The images content is still meaningful and was not accidentally corrupted during the process

4 CNN from scratch

4.1 2 class

The task consists in the development of an ad-hoc CNN to distinguish between masses and calcifications in mammograms. It is notoriously hard to predict the performance of a neural network a priori, and a model that works particularly well in one task may be useless elsewhere. Therefore the only viable approach to find a good layout is by trial and error, with the aid of guidelines and 'best practices' suggested by experienced researchers.
The first step is to understand how large the model should be. A network that is too small won't be able to generalize well; on the other side, a model with too many parameters may learn slowly and overfit. A good way to find the appropriate size is to start from a small naive model, that is likely to underfit; then, gradually increase its size until it begins overfitting, which roughly indicates that the network generalization capability is no longer limited by the number of parameters. The network can be refined thereafter, adjusting the size of individual layers, tuning the hyperparameters, including regularization techniques and so on.
During the training, the validation loss must be constantly monitored to spot signs of overfitting; also, the model (namely its weights) associated to the lowest validation loss is stored and used as reference for the subsequent performance analysis. Why to observe the validation loss rather than validation accuracy? The former is more informative: while the accuracy represents just the fraction of correctly classified samples over the total, the loss also measures how confident the network was in making the predictions.

4.1.1 Experiment 0 - Simple model

The first model is a very small CNN, made of 2 convolutional layers interleaved with max-pooling. At the end, after a fully-connect layer, a single neuron with sigmoid activation generates the output.

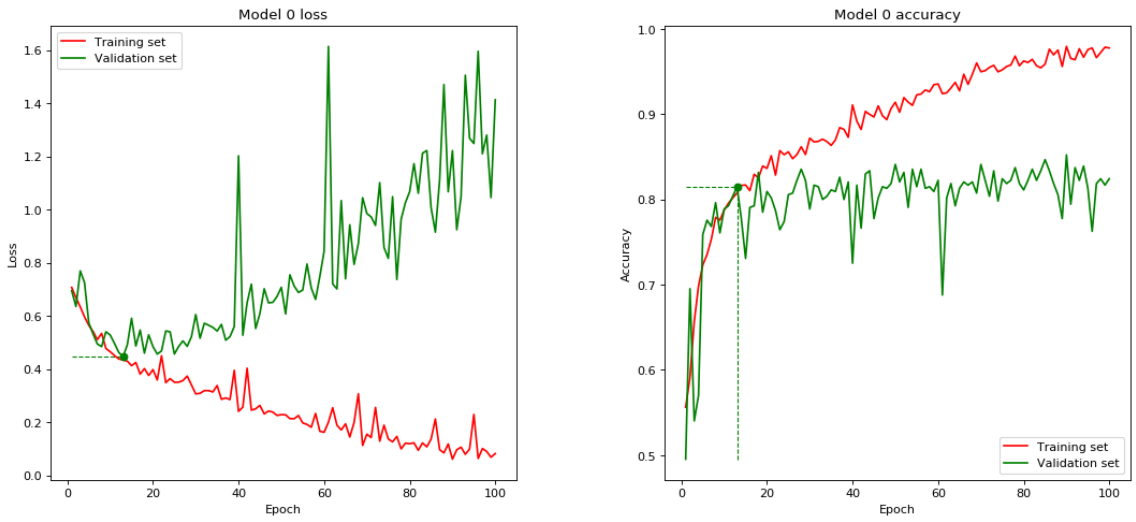
```
Conv2D(32, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(64, (3, 3), relu)
MaxPooling2D((2, 2))
Flatten()
Dense(16, relu)
Dense(1, sigmoid)
```

The aim is to get a rough idea of the needed model complexity. It is worth to say a few words about the optimization procedure. The most appropriate loss function here is the *binary crossentropy*, since the problem is a binary classification one.

$$J(w) = \frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

where y_i denotes the true label and $\hat{y}_i$ the predicted one.
The optimizer is *RMSprop*, an adaptive algorithm proposed by G. Hinton, considered by many a reliable and safe pick in most of the situations. Parameters like learning rate and decay are initially left with their default values, and will be tuned later (see Section 4.2).
The batch size is set to 32, that is the number of samples considered in every iteration. Mini-batching is a tradeoff between two extremal approaches: *SGD (Stochastic Gradient Descent)* (one example at a time, converges fast but is noisy) and *Batch Gradient Descent* (all the data, smooth). A larger batch requires more memory; though, GPUs are often able to parallelize the evaluation of samples within the same batch.
After training for 100 epochs, the results are shown in Fig. 1 and listed below. The summary contains the loss and accuracy values on both the validation and testing sets, evaluated in correspondence of the validation loss minimum.

Experiment 0				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
13	81.50	77.38	0.4460	0.5247



(a) Loss (b) Accuracy

Figure 1: Scratch CNN 0

The network learned relatively fast, and eventually reached an almost perfect classification on the training dataset, which of course does not make it good at classifying other data. Indeed, the accuracy achieved on the validation set was 81.5%, with a 77.4% on the test set. Indeed, the model severely overfits, mainly due to the limited dataset (only 2000 images for 1.3 million of weights) and lack of regularization methods. Nonetheless, it is a promising initial result and a good starting point for subsequent models. This experiment proves that very deep architectures are not strictly needed in this task to achieve good results. Clearly, larger and deeper models may help in the classification of more difficult samples, it is yet to be proven.

4.1.2 Experiment 1 - Dropout

The priority now is to mitigate the hard overfitting observed in the previous experiment, which prevents the network from finding a good general set of weights.
Model 1 is just like the previous one, but with the addition of a Dropout layer after the final fully-connected block. *Dropout* is a powerful regularization technique, patented by Google, that is often employed to reduce overfitting [24]. It consists in ignoring a randomly chosen fraction of the previous layer neurons at every iteration in the training phase; this encourages the network to find an alternate and redundant representations for the inputs, ultimately leading to better performance. Dropout is generally avoided in convolutional layers because it provides little to no benefit there, probably because of the intrinsic redundancies of convolutional neurons. Nevertheless, some authors have studied in details the effect of dropout on conv layers, proposing ad-hoc variations of dropout too [15].
On the other side, on fully-connected layers, a dropout rate of 0.5 is considered an optimal choice by the dropout inventors themselves [24].

```
Conv2D(32, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(64, (3, 3), relu)
MaxPooling2D((2, 2))
Flatten()
Dense(16, relu)
Dropout(0.5)
Dense(1, sigmoid)
```

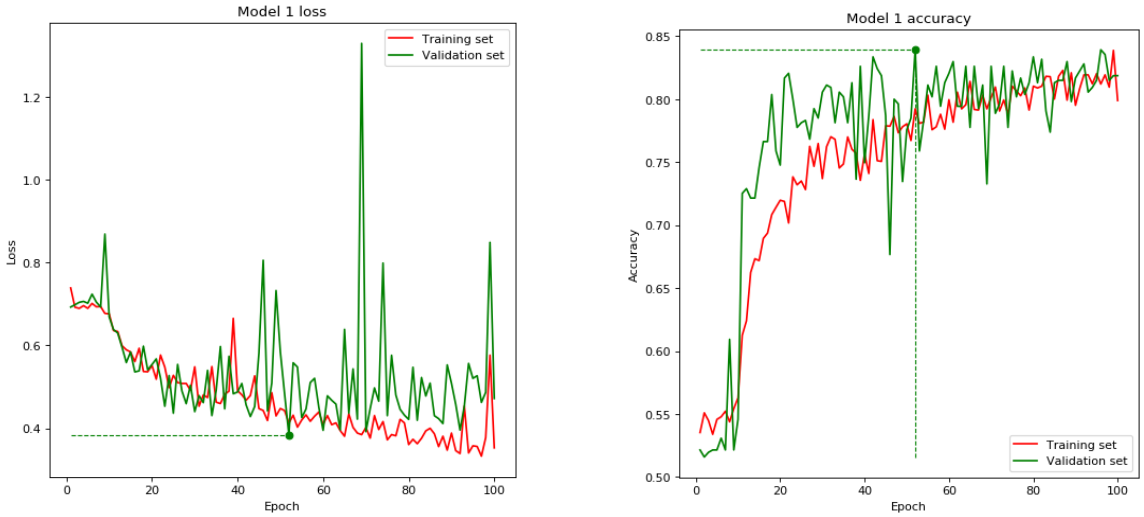
After training for 100 epoch, these are the results:

Experiment 1				
Epoch optimal	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
52	83.93	80.65	0.3837	0.6088

The only addition of a Dropout layer greatly reduced the overfitting w.r.t the previous experiment, without completely eliminating it though. Now that the effect of overfitting was less disruptive, the network managed to improve its accuracy as well.

4.1.3 Experiment 2 - Data augmentation

Before changing the architectural layout, another regularization method is applied to the previous model to further mitigate overfitting: *data augmentation*. With data augmentation, the network is trained with a higher number of different samples, generated by applying predefined transformations to the original data. Common techniques are image flipping, shifting, rotation, scaling, distortion, noise injection. Data augmentation basically extends the original dataset with more images; although being correlated, these pictures still introduce noise and variance in the training set, reducing the chance of overfitting.



(a) Loss (b) Accuracy

Figure 2: Scratch CNN 1

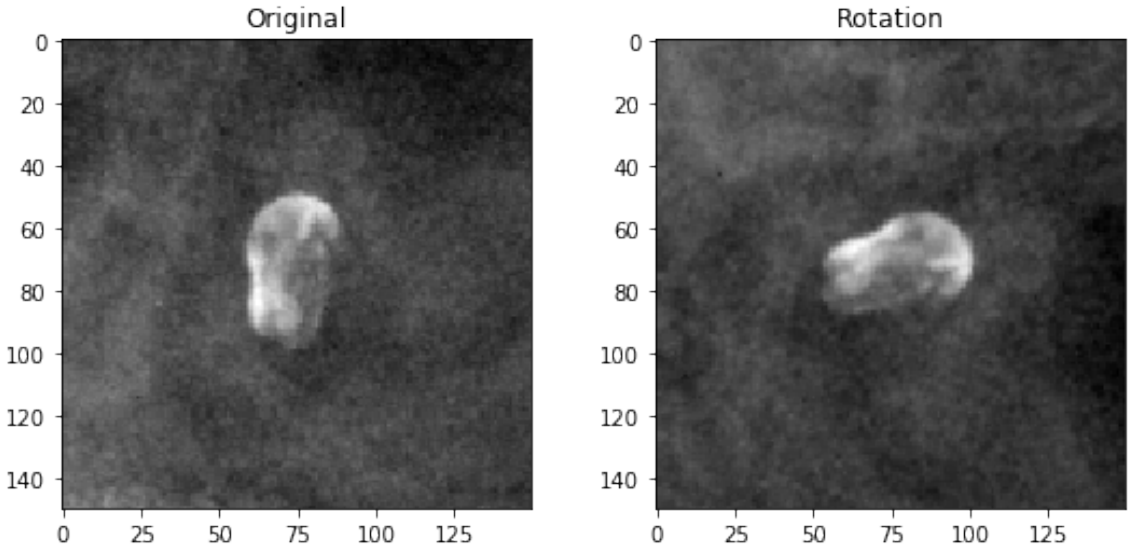
Like in other studies on the same dataset [3][7][10], here I use these methods:

- Flip (horizontal and vertical)
- Rotation (angle between 0 and 180 degrees)
- Shear (10 deg)
- Scale (20%)

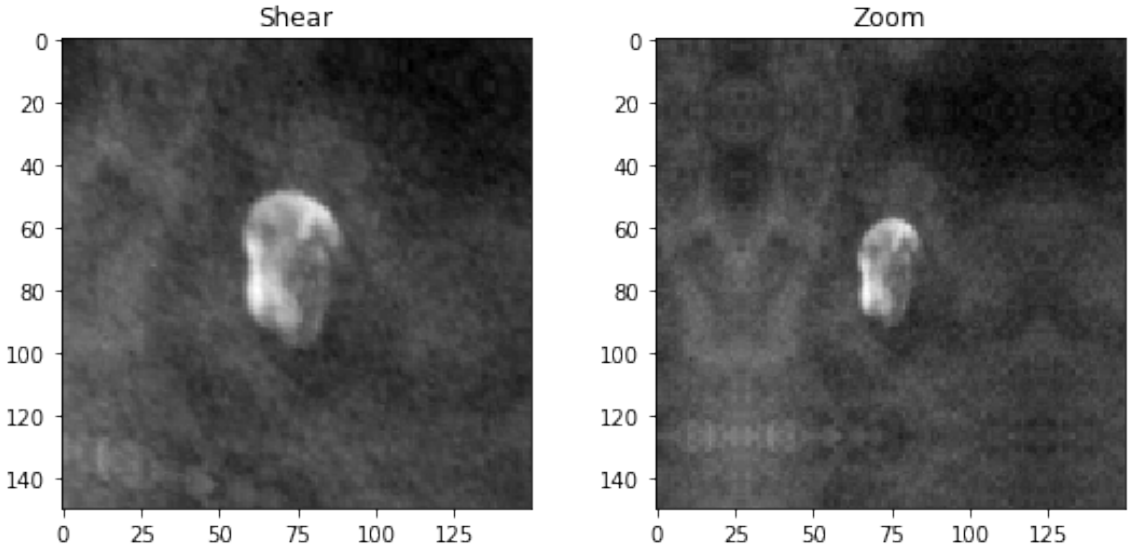
The effects of data augmentation are illustrated in Figure 3. The network architecture remains unchanged. The results are shown in Figure 1.

Experiment 2				
Epoch optimal	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
87	80.56	79.76	0.4458	0.4999

Thanks to data augmentation, overfitting is no longer present, as both training and validation loss decreased over time (with some noise). The accuracy slightly dropped because augmentation makes the learning task harder by design, however the network kept learning during the whole run, as the training accuracy suggests; actually, after 100 epochs, there was still room for improvement. In the future experiments, a smarter strategy will be adopted to choose the number of epochs, to avoid halting the training too early. Two more considerations can be made. First, the optimization process was quite noisy, which could be a symptom of a too small batch size. Second, the validation accuracy was consistently greater than the training one, although the network did not learn from the validation images. This benign effect is often caused by dropout: that layer is indeed disabled when evaluating the accuracy on the validation set, so the output neuron can exploit more information thus producing a more reliable result.



(a) Original (b) Rotation

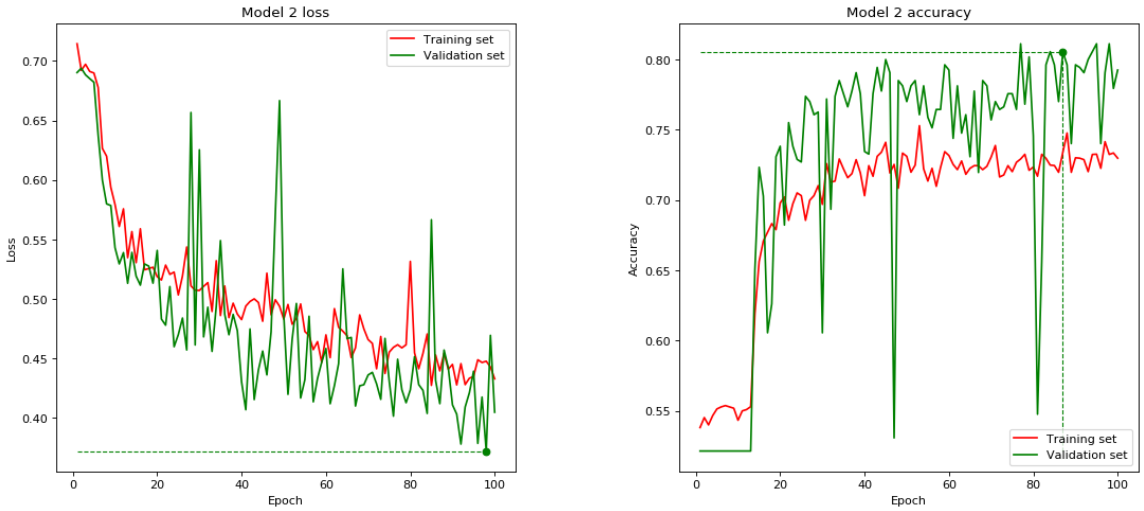


(c) Shear (d) Zoom

Figure 3: Effects of data augmentation

4.1.4 Experiment 3 - Larger network, Early stopping

Model 3 introduces some changes both in the model architecture and the optimization process. With respect to the previous model, it features one more convolutional block. The number of filters is 128, so the information volume drop remains limited. The fully-connected layer now has twice more neurons (32), to cope with the increased filters. The batch size is enlarged to 128 (4x) to achieve a smoother learning curve. Increasing the batch size is a practical alternative to lowering the learning rate, as proposed by Smith et al., which also offers the advantage of increased paral-



(a) Loss (b) Accuracy

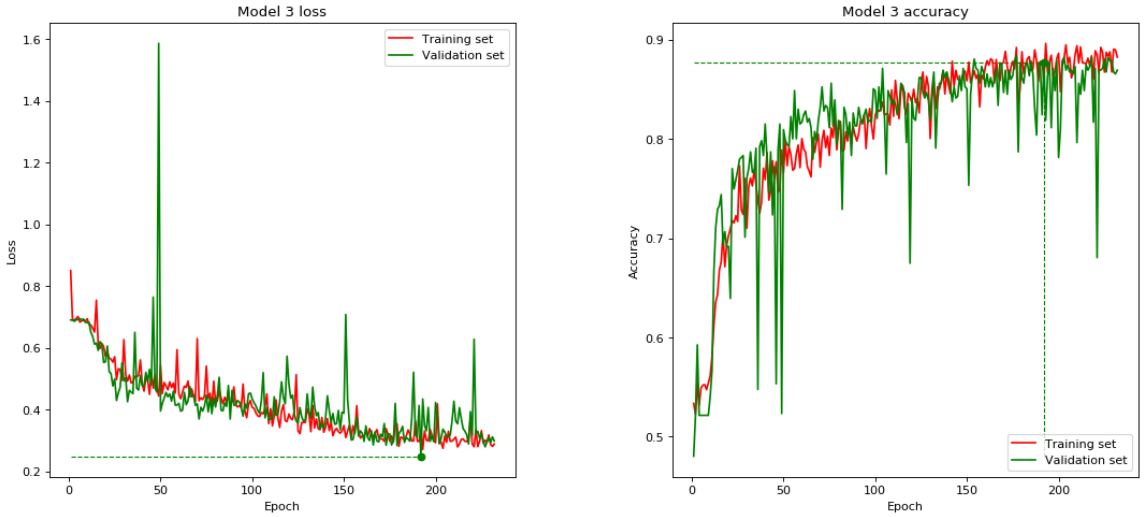
Figure 4: Scratch CNN 2

lelism [23]. Since it is not possible to know a priori how long it will take to converge towards the optimum, the network is allowed to learn for a very high number of epochs, until the validation accuracy stops decreasing or drops due to overfitting. This is called *early stopping*, and consists in interrupting the training if the network didn't make any progress recently. The validation loss is a simple and effective metric to evaluate the quality of a model over time, so this experiment will use it to quit training. It should be noted, though, that more elaborate strategies have been proposed for early stopping when the validation loss is noisy [16].
It is worth to remember that training time is another important parameter to judge the quality of a NN, and sometimes the researchers make a trade-off between accuracy and training time; that said, this task deals with medical images, where the accuracy must be as high as possible for obvious reasons, and the time needed for training becomes a secondary factor.

```
Conv2D(32, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(64, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(128, (3, 3), relu)
MaxPooling2D((2, 2))
Flatten()
Dense(32, relu)
Dropout(0.5)
Dense(1, sigmoid)
```

Early stopping allowed the model to train for 232 consecutive epochs, with the validation loss minimum reached at epoch 192. The results are shown below:

Experiment 3				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
192	87.66	85.42	0.2489	0.3645



(a) Loss (b) Accuracy

Figure 5: Scratch CNN 3

Overall the network scored a 85.4% accuracy on the test set and 87.7% on the validation one, a big improvement compared to previous models. Although the training is still noisy, overfitting is not a major issue anymore. In this and the previous examples the testing accuracy turned to be consistently lower than the validation one. This could be either due to a statistical fluctuation (1% means as few as 3 images), or because the testing set contains slightly more challenging samples than the validation one.

4.1.5 Experiment 4 - Larger FC

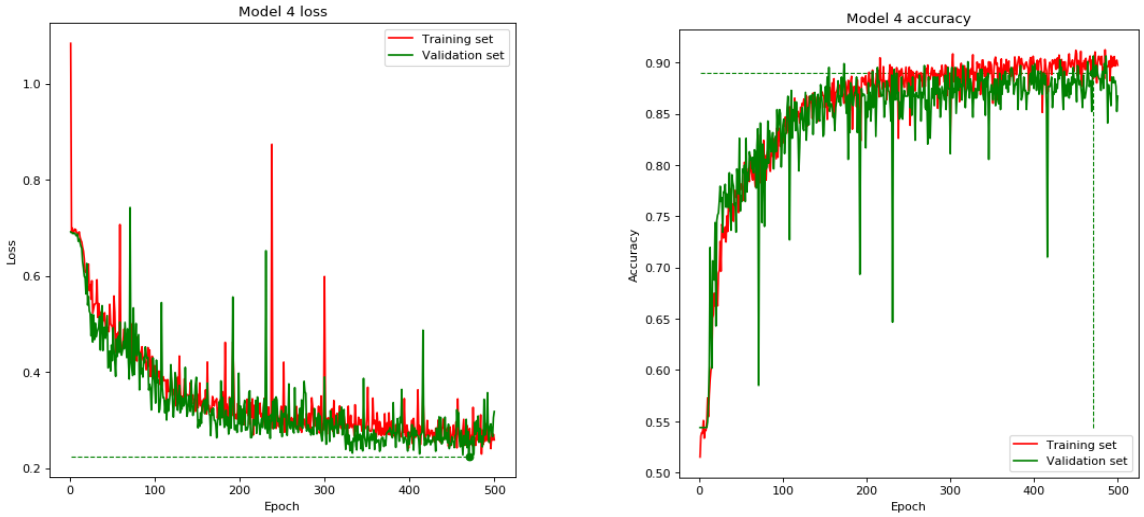
A good thing to verify is whether the fully-connected layer size, where most of the weights are located, represents a bottleneck. Therefore, Model 4 evolves from Model 3 by simply increasing to 48 the number of neurons in the fully-connected layer. As a side tweak, the early-stopping patience has been double because positive fluctuations seem to occur every 40-50 epochs on average, based on the previous graph. The updated architecture is:

```
Conv2D(32, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(64, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(128, (3, 3), relu)
```

```
MaxPooling2D((2, 2))
Flatten()
Dense(48, relu)
Dropout(0.5)
Dense(1, sigmoid)
```

And here are the results:

Experiment 4				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
470	88.97	87.50	0.2251	0.3332



(a) Loss (b) Accuracy

Figure 6: Scratch CNN 4

Increasing the FC layer size positively affected the network performance (2% accuracy gain w.r.t. the previous model), eventually achieving a solid 87.5% accuracy on the test set. This result is in-line with the accuracy achieved by state-of-the-art architectures like VGG16 and ResNet50, as confirmed in this (see Section 5.1.1) and other studies [1][27].

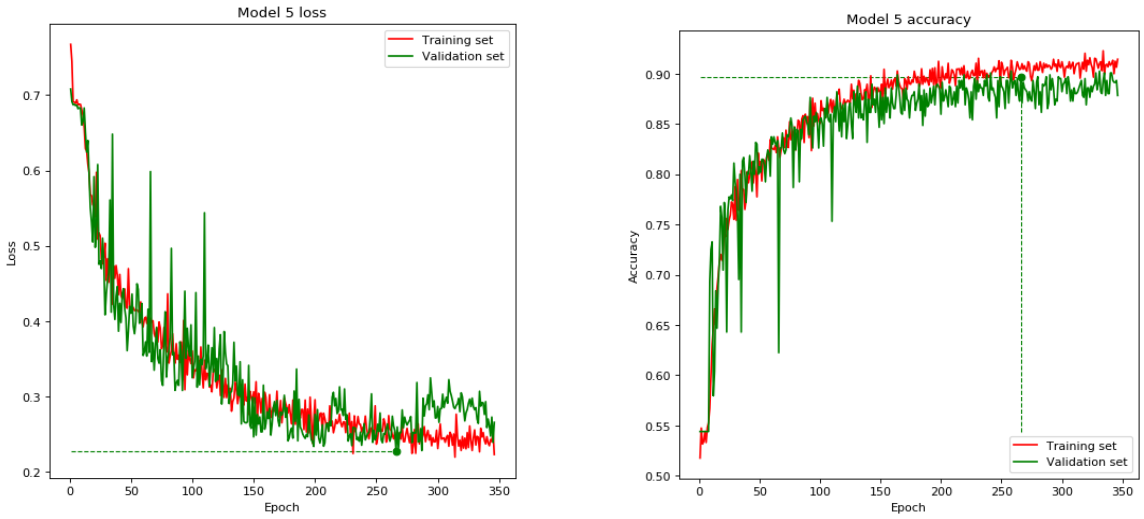
4.1.6 Experiment 5 - Learning rate decay

One problem of the previous experiments was the noisy loss and accuracy history. The large fluctuations occur because the weight updates are relatively large, even when the global minimum is very close. This degree of freedom is not a malus per se, especially in the early phase, when the exploration helps avoiding local minima. Model 5 attempts to balance exploration and exploitation by the mean of a *learning rate decay* factor, which reduces the learning rate over time. The decay formula is:

$$LR_n = \frac{LR_0}{1 + \gamma n}$$

where n is the number of batch updates performed so far, and γ is the decay factor. The latter was set to 0.001, so that the LR decreases by an order of magnitude every 115 epochs approximately (20 batch updates per epoch).
After a long training, the network produced the following results:

Experiment 5				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
266	89.72	88.69	0.2282	0.3048



(a) Loss (b) Accuracy

Figure 7: Scratch CNN 5

As expected, the oscillations greatly reduced as a consequence of the learning rate decay. The convergence time is still long, but the testing accuracy increased to almost 89%. It is possible to observe a moderate overfitting at the end, caused by the fact that smaller steps make the loss minimum easier to reach.

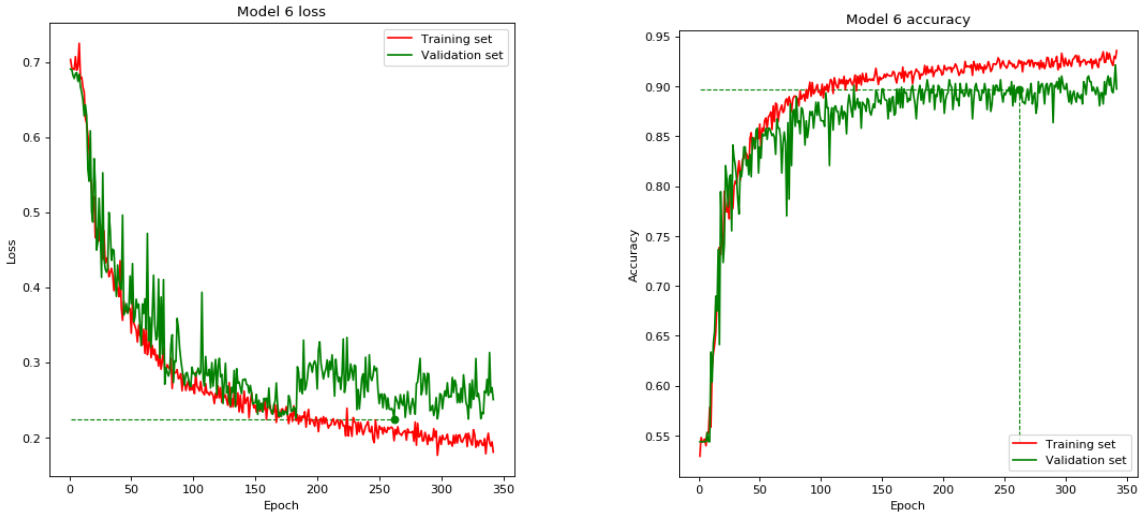
4.1.7 Experiment 6 - Deeper

Models 6 is like the previous one, with the addition of an extra convolutional block with 256 features. Moreover, the learning decay rate has been reduced by a factor of 10 to prevent an excessively long training time.
The purpose of this experiment is to verify if a deeper convolutional stage helps improving the accuracy or not. This strongly depends on the nature of the problem; in general, it is strongly related to the kind of object to classify. Sometimes it works, sometimes it does not.

```
Conv2D(32, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(64, (3, 3), relu)
```

```
MaxPooling2D((2, 2))
Conv2D(128, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(256, (3, 3), relu)
MaxPooling2D((2, 2))
Flatten()
Dense(48, relu)
Dropout(0.5)
Dense(1, sigmoid)
```

Experiment 6				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
262	89.72	87.80	0.2242	0.3449



(a) Loss (b) Accuracy

Figure 8: Scratch CNN 6

The addition of another convolutional block reduced the overall number of parameters by a factor 2, without worsening the final accuracy. A smaller model is always a positive thing, as it becomes easier to share and often requires less memory. It is also interesting to point out that the network started overfitting.

4.1.8 Experiment 7 - Larger kernel

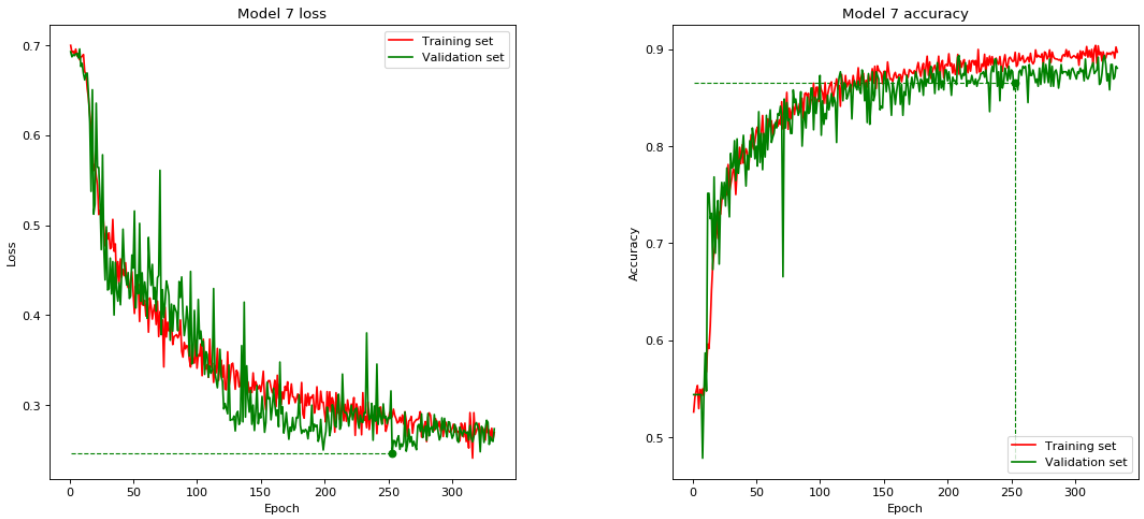
Another important parameter that affects the performance of a CNN is the kernel size. A convolution performed by the means of a larger kernel is able to capture details at a larger scale, which can be beneficial in some situations. Usually, when a CNN uses filters larger than 3x3, the layers are arranged so that larger kernels precede smaller ones, to progressively narrow the receptive field.

Model 7 is very similar Model 4, with one modification: the first conv2D layer now has a 5x5 kernel with stride 2.

```
Conv2D(32, (5, 5), stride=2, relu)
MaxPooling2D((2, 2))
Conv2D(64, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(128, (3, 3), relu)
MaxPooling2D((2, 2))
Flatten()
Dense(48, relu)
Dropout(0.5)
Dense(1, sigmoid)
```

Note how the changes were minimal and only in the first layer: the assumption is that, if having a larger kernel is beneficial, then a small accuracy improvement should be observed even after a small change like this. The stride was incremented to avoid excessive overlapping between the receptive fields of adjacent neurons.

Experiment 7				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
253	86.54	87.50	0.2473	0.3190



(a) Loss (b) Accuracy

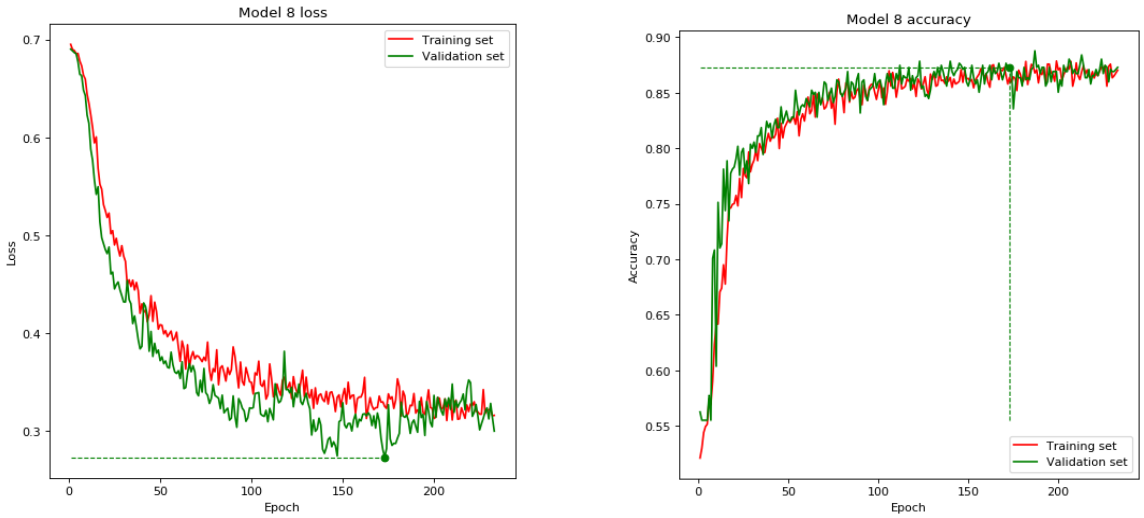
Figure 9: Scratch CNN 7

Using a larger kernel size does not help in this specific classification problem. In fact, the accuracy did not improve w.r.t. to models with 3x3 filters.

4.1.9 Experiment 8 - Adam

So far all the experiments were conducted using RMSprop, which a well-respected optimizer but not necessarily the best choice in every situation. Another popular optimization algorithm is *Adam*, an extension of the stochastic gradient descent. As showed in Section 4.2, Adam may converge faster than RMSprop when the learning rate is properly tuned. Moreover, some researchers got relevant results using Adam on DDSM [10]. The next experiment recycles the architecture of Model 4, but the training is performed with Adam here.
The results are plotted in Figure 10.

Experiment 8				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
173	87.29	81.55	0.2735	0.3985



(a) Loss (b) Accuracy

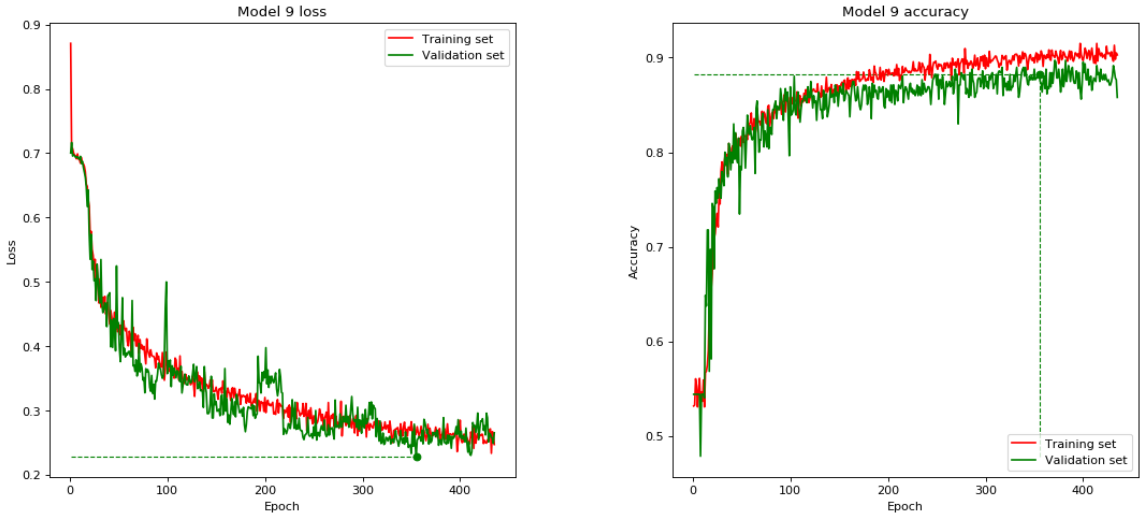
Figure 10: Scratch CNN 8

Compared to RMSprop, Adam oscillates too but without hard spikes; the accuracy graph appears more stable indeed. The downside is in the accuracy itself: although it grows quite fast in the early phase, it settles below the RMSprop level, and the results achieved on the testing set confirm this.

4.1.10 Experiment 9 - L2 regularization

This experiment aims at removing the overfitting seen in Model 6, by means of another well-known regularizer: *L2*. A small coefficient was chosen to avoid an excessive regularization (that would lead to worse performance), since we already use other techniques like dropout and augmentation.
The results are shown below in Figure 11.

Experiment 9				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
355	88.22	88.10	0.2287	0.3329



(a) Loss (b) Accuracy

Figure 11: Scratch CNN 9

L2 regularization successfully managed to removed overfitting, so that the validation loss finally matches the training one with no significant deviation.

4.1.11 Experiment 10 - Batch normalization

The last experiment aims to test the effects of *batch normalization*, a popular technique used to improve the speed, performance and stability of a network. It consists in normalizing (subtract mean and divide by variance) the inputs of each layer for each mini-batch. The optimal position of batch normalization w.r.t. the activation layer is still debated: although the paper authors used to put it before the activation to avoid altering the output distribution, many researchers have obtained better results by placing it after the Relu [13]. The final layer is usually excluded from batch normalization, as non-normalized inputs typically help predicting the correct class. Here follows the model used for this experiment:

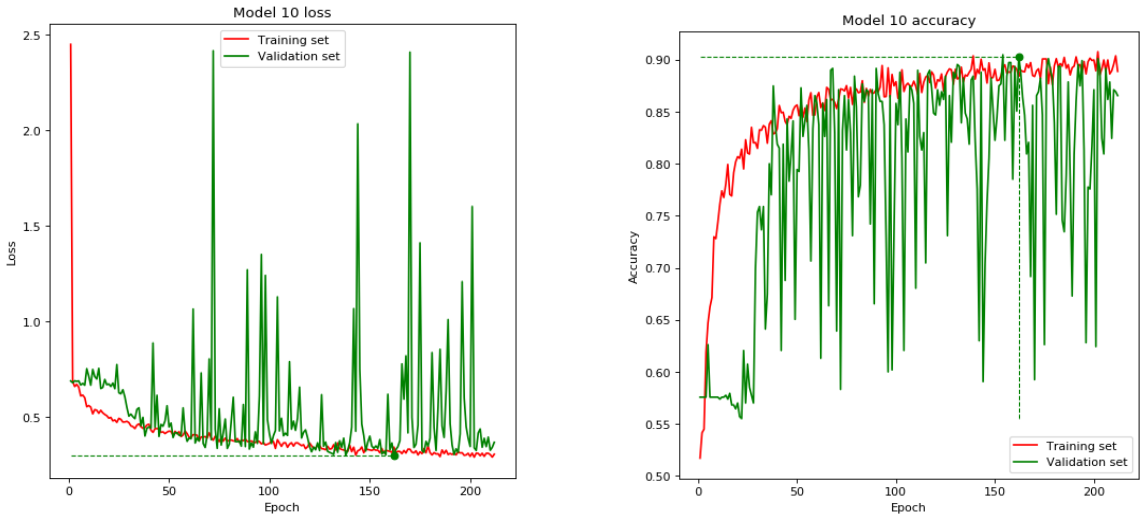
```
Conv2D(32, (5, 5), stride=2)
BatchNormalization()
Activation(relu)
MaxPooling2D((2, 2))
Conv2D(64, (3, 3))
BatchNormalization()
Activation(relu)
```

```
MaxPooling2D((2, 2))
Conv2D(128, (3, 3))
BatchNormalization()
Activation(relu)
MaxPooling2D((2, 2))
Conv2D(256, (3, 3))
BatchNormalization()
Activation(relu)
MaxPooling2D((2, 2))
Flatten()
Dense(48, relu)
Dropout(0.5)
Dense(1, sigmoid)
```

The results achieved by the CNN with batch normalization are summarized here.

Experiment 10				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
162	90.28	85.42	0.2996	0.4538

It is easy to see that batch normalization causes wild oscillations in validation loss and accuracy. Since the BN hyperparameters were left to their default values, this behaviour is probably due to an excessive momentum.



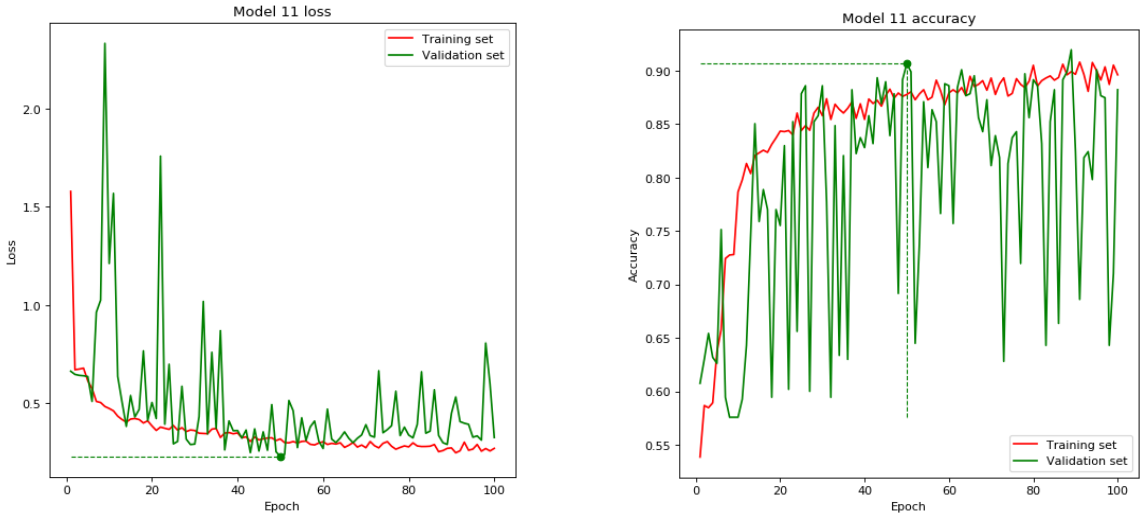
(a) Loss (b) Accuracy

Figure 12: Scratch CNN 10

4.1.12 Experiment 11 - BN (smaller momentum)

This is a replica of the previous experiment, but with a much smaller batch normalization momentum: 0.001.

Experiment 11				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
50	90.65	85.12	0.2304	0.4099



(a) Loss (b) Accuracy

Figure 13: Scratch CNN 11

Lowering the momentum decreased the large fluctuations, especially in the final stage. The convergence was faster, and the loss consistently below the average w.r.t. the previous experiments. Nevertheless, the final accuracy did not improve significantly. Overall, batch normalization seems to be not particularly beneficial in this specific classification task.

4.2 Learning rate analysis

This section describes a procedure to find an appropriate learning rate for NN models, based on a LR-update strategy proposed by Smith [22].

4.2.1 Estimation

The experiment involves four of the most popular optimizers for NNs: SGD, RMSprop, Adam and Nadam [18]. In order to roughly approximate the range of reasonable LR values for each optimizer, a simple strategy is adopted: starting with a very low LR, its value is slightly increased at the end of each epoch. Initially, the network will learn slowly, because a small LR does now allow large weight updates, hence the loss will remain more or less constant. Then, the LR increases and the loss starts decreasing. At some point, however, the learning rate becomes so big that updates cause large and unpredictable fluctuations of the loss, and the network basically starts diverging. In practice, the loss will start from a value around 0.69, corresponding to random prediction. After it gets lower a certain threshold, let's say

0.6, one can safely assume that the network started learning, and will eventually reach a loss minimum. Later, as soon as the weights diverge and the network goes back to random prediction, the training is stopped.

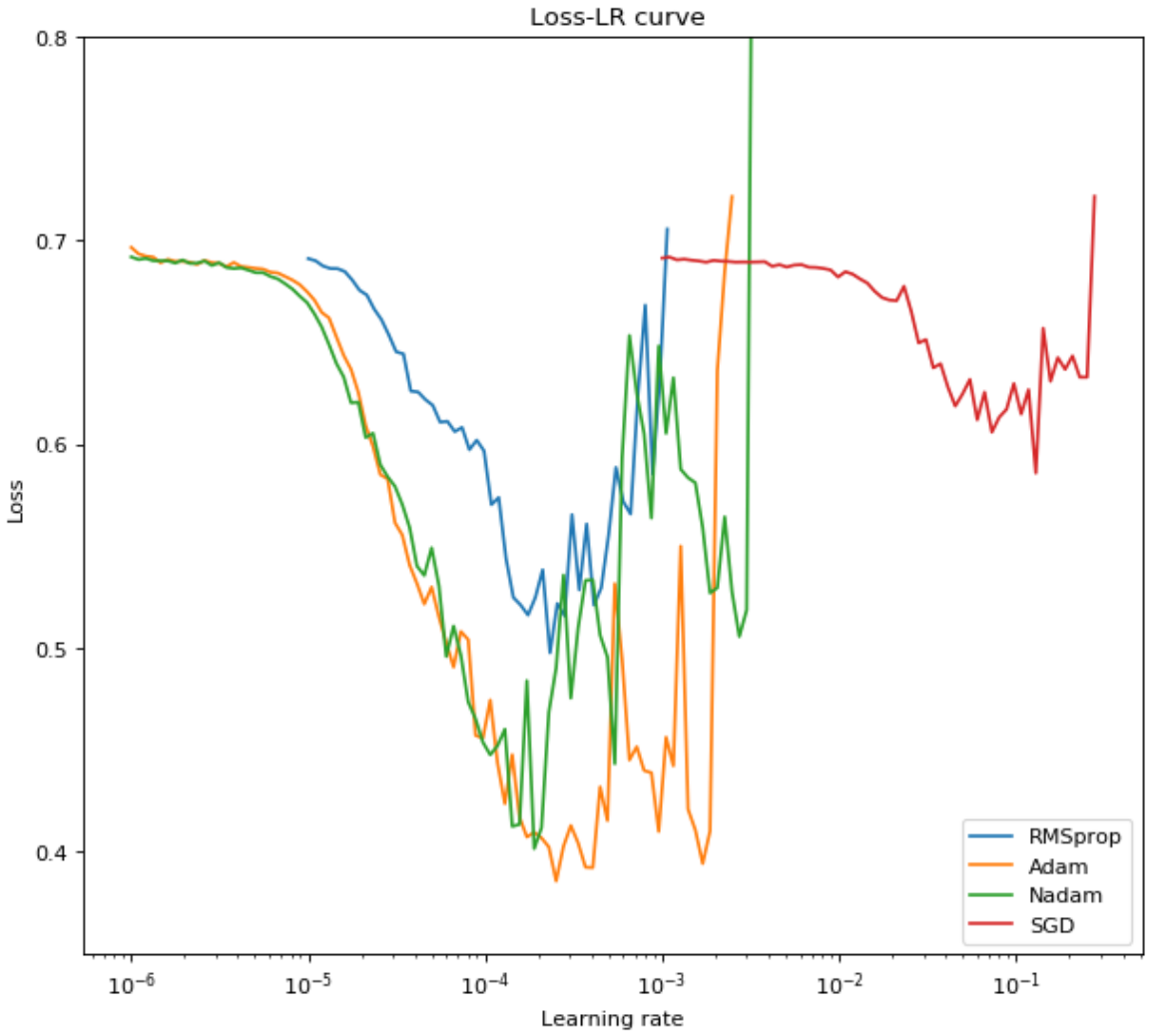


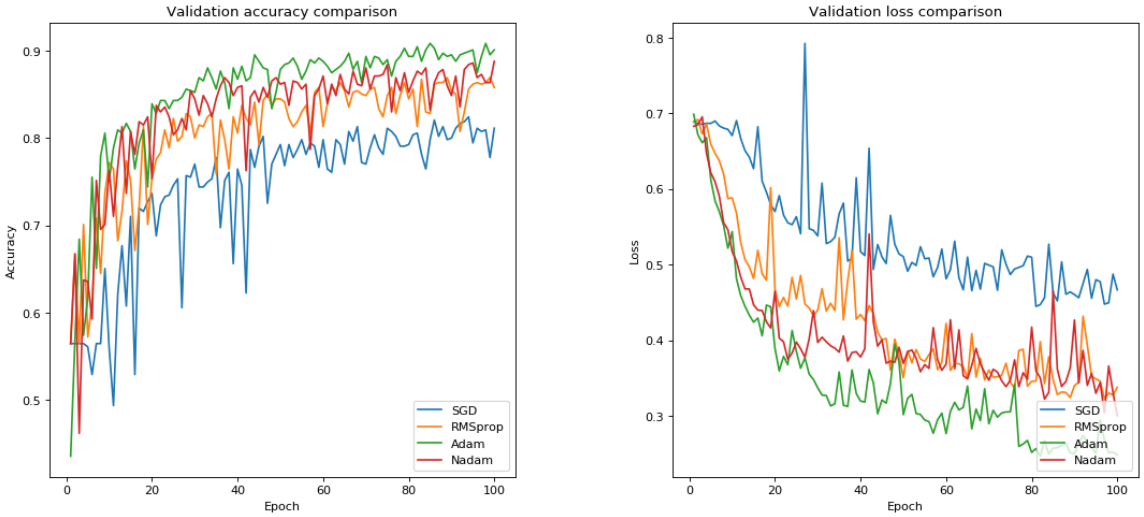
Figure 14: LR-loss curve for different optimizers

The graph above clearly shows that learning rate plays a decisive role during the training. When it is too high, weight updates become too large and the network becomes unstable, failing to converge towards the loss minimum. On the other hand, if it set too low, the network learns slowly and we observe modest improvements between two consecutive epochs. The global minimum of the LR-loss curve indicates the point where the learning rate starts causing instabilities, hence choosing a greater value is discouraged. Ideally, the best one is in the region with the fastest descent of loss function, that is where the plotted curve is steepest (negatively). It should be also noted that, in a stable network, loss variations naturally decrease over time, even if the LR remains constant, as a consequence of the gradual convergence of the weights towards the optimum. Thus, the steepest point represents may not directly represent the optimal LR, but a lower bound for it. That said, a practical way

to choose an adequate LR for an optimizer is to pick a value between the steepest point and the minimum, e.g. in the middle of this region.
In this case, reasonable choices are:

- SGD : 3e-2
- RMSProp : 1e-4
- Adam : 1e-4
- Nadam : 1e-4

Note how these values slightly differ from the Keras default ones.
Running each optimizer for some epochs (let's say 100) with these learning rates is useful to get a (rough) preview of each optimizer behaviour.



(a) Loss (b) Accuracy

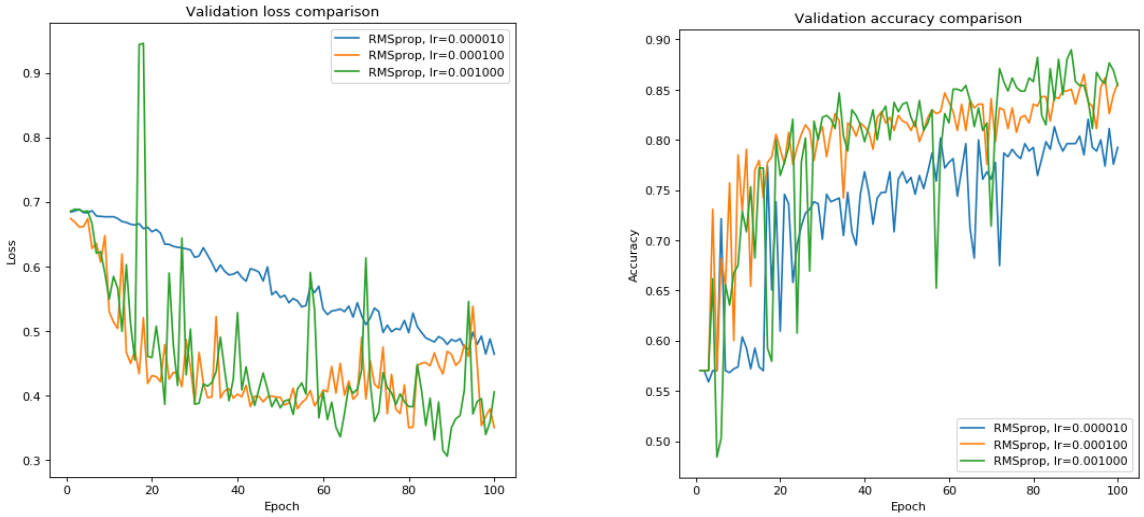
Figure 15: Optimizers comparison

The graphs shows that SGD is relatively weak with respect to the other optimizers. Adam converges faster than RMSprop and Nadam, whose curves are quite similar. Only RMSprop and Adam will be considered from now on, since SGD is clearly dominated while Nadam is nothing but an enhanced Adam.

4.2.2 Verification

Now that approximate values for the learning rate have been discovered, one may try to directly experiment with different nearby values and find which is better.

RMSprop
A good LR value for RMSprop was estimated to be 1e-4, thus the next experiment compares it with 1e-3 (10x larger) and 1e-5 (10x smaller). Each test is run independently from the others, for 100 epochs.



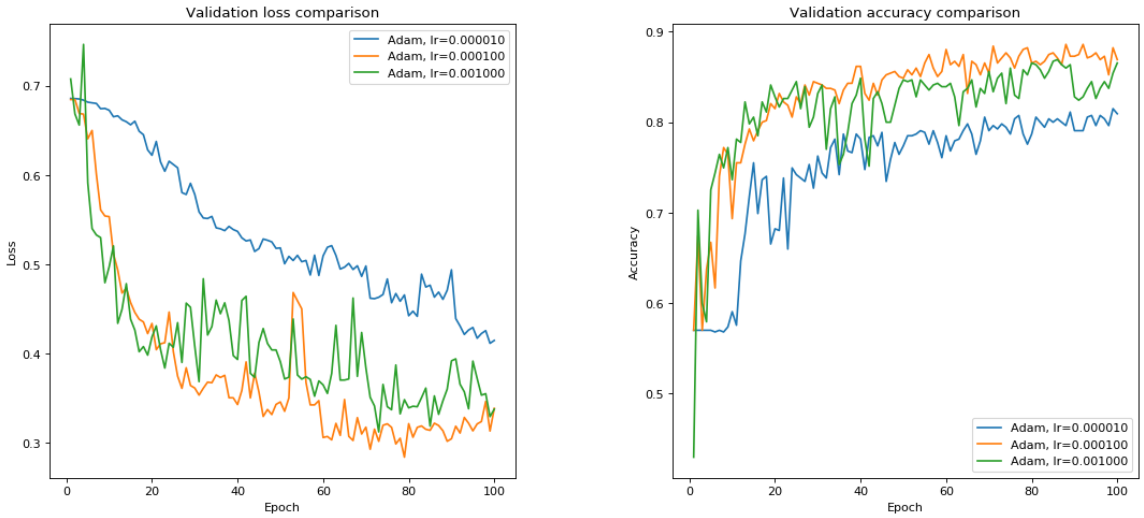
(a) Loss (b) Accuracy

Figure 16: RMSprop learning rate comparison

According to Figure 16, values between 1e-4 and 1e-3 produce similar results. 1e-3 is more noisy, but converges a bit faster. On the other hand, 1e-5 represents an excessively low value.

Adam

Just like RMSprop, Adam optimal value is to be found in the range (1e-5 - 1e-3).



(a) Loss (b) Accuracy

Figure 17: Adam learning rate comparison

Based on Fig. 17, 1e-5 is definitely a bad choice for the learning rate, as it makes the network converge very slowly. Interestingly, 1e-4 produces better results than 1e-3.

4.3 Error analysis

Let's take a look at the results obtained by the previous experiments, especially analyzing the misclassified samples. The aim is to shed some light about these images, and understand why the classifiers predicted the wrong class. Three (independently trained) models are selected among the best performing ones, according to the previous experiments: Model 5, 6 and 9.
In the following code the term 'misprediction' indicates any sample that led to a wrong label prediction; 'hard misprediction' denotes the mispredictions whose output value differs more than 0.8 from the expected one (e.g. predicted 0.1 when the actual label was 1), like in Figure 18.

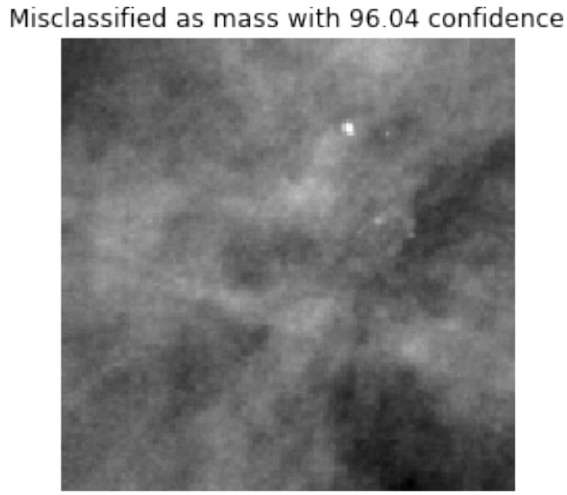


Figure 18: Example of a misclassified calcification sample

Analyzing the predictions of the three models, some interesting considerations arise:

```
5 and 6 differ for 33 samples
5 and 11 differ for 24 samples
9 and 11 differ for 23 samples
23 samples were misclassified by all three
7 samples were hard misclassified by all three
```

All the classifiers made some mistakes, but interestingly there are samples that were correctly labeled by one classifier and incorrectly by another. In other words, even if the accuracy scores are similar and their architectures are similar as well, they behave very differently.
Two dozens of images (approximately 7% of the total) were labeled as belonging to the wrong class by all three classifiers: medically speaking, these are likely to be hard samples to diagnose, where the abnormality appears atypical. On the other side, 93% of the mammograms were correctly classified by at least one model. The fact that different models produce different results can be exploited to build an ensemble, hopefully able to achieve a greater accuracy than the individual classifiers.
Figure 19 shows the three ROC curves for models 5, 6 and 9. They intersect multiple times, hence none is definitely better than the other; depending on where the

threshold separating masses from calcification is set, the three models behave differently. One may get a better sensitivity, another a better specificity instead. A metric that is often use to prefer one classifier over another is the AUC (Area Under Curve, the bigger the better): here, model 5 has the largest area (0.954), while Model 6 has the smallest one (0.938).

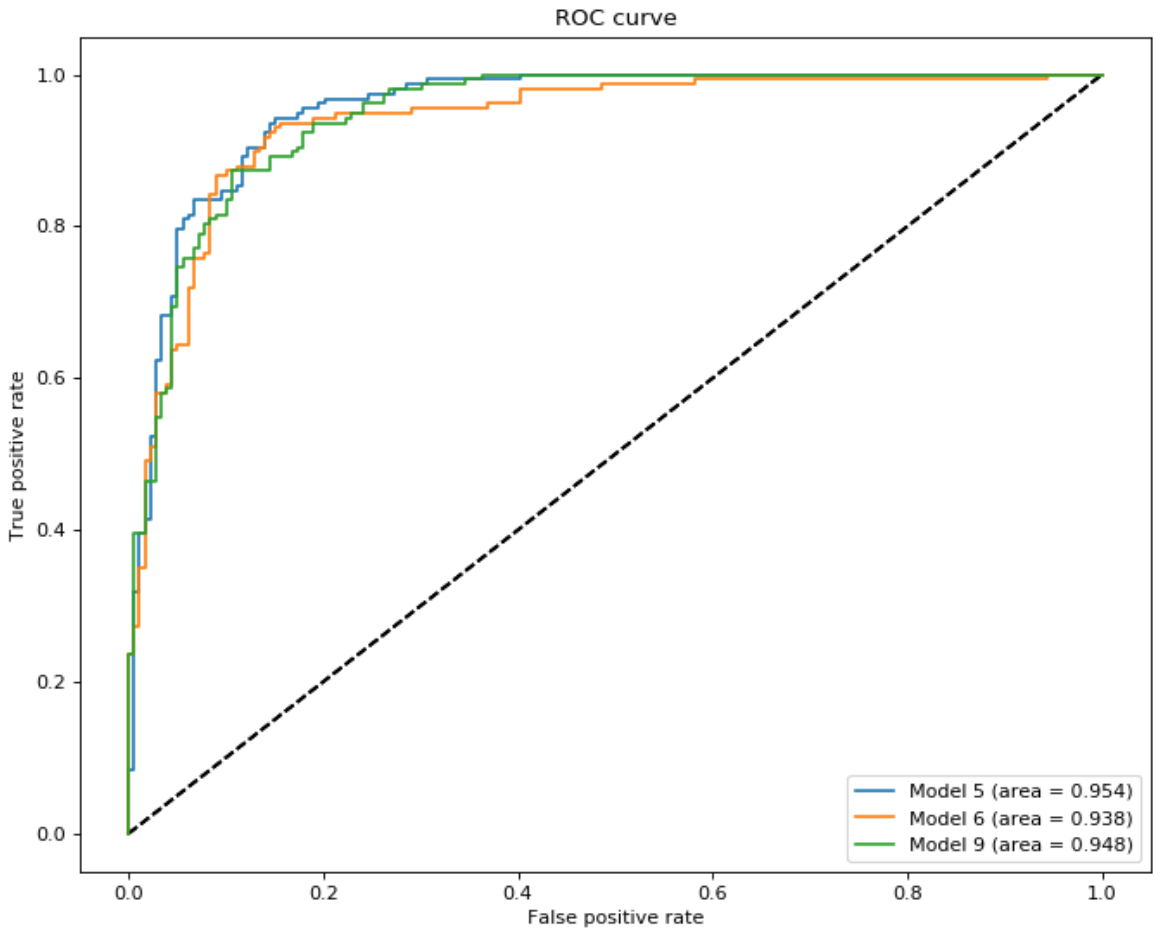


Figure 19: ROC and AUC comparison for Models 5,6 and 9

4.4 4 class

The task is similar to the previous one, but instead of distinguishing only between masses and calcifications there is an additional challenge: predict whether the abnormality is benign or malignant.

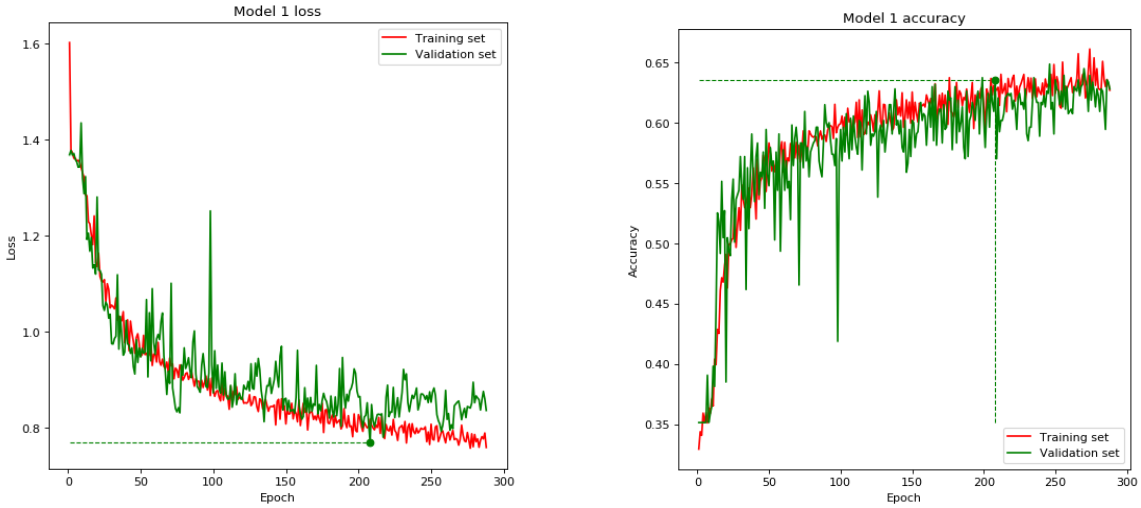
4.4.1 Experiment 1 - Previous best

The model that achieved the best performance in the two-class classification is a good starting point for sure, since both tasks partially overlap and they work on the same dataset. That CNN had an accuracy above 88% on the test set. Of course, some modifications are required, especially in the output layer, because the problem is now a multiclass classification: 4 softmax neurons replace the old sigmoid unit. Likewise, the loss function is now the categorical crossentropy, a generalized version of binary crossentropy.

```
Conv2D(32, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(64, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(128, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(256, (3, 3), relu)
MaxPooling2D((2, 2))
Flatten()
Dense(48, relu)
Dropout(0.5)
Dense(4, softmax)
```

Experiment 1				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
208	63.55	57.74	0.7709	0.9402

The network obtained a 57.7% final accuracy on the test set. This value is significantly lower than the almost 88% achieved in the 2-class classification task, but it is an obvious consequence of the increased number of classes. Now it is sufficient to mispredict the type OR the malignancy to consider that sample classified wrong. The confusion matrix provides very useful information here. Remembering how the labels were mapped to classes , one may observe that 44 images were assigned the wrong type (mass/calc), that is corresponds 13.1% of the samples. In other words, ignoring the malignancy, that corresponds to a 86.3% accuracy, in line with the results achieved by the pure 2-class classifier. It is worth to point out that most of the mistakes are indeed due to benign/malignant misclassification, rather than mass/calcification swapping. Apparently, the task of predicting the harmfulness is inherently a more difficult task than identifying the type of abnormality. Finally, a moderate overfitting is noticeable at the end of the training.



(a) Loss (b) Accuracy

Figure 20: Scratch CNN 1 (4-class)

4.4.2 Experiment 2 - More filters

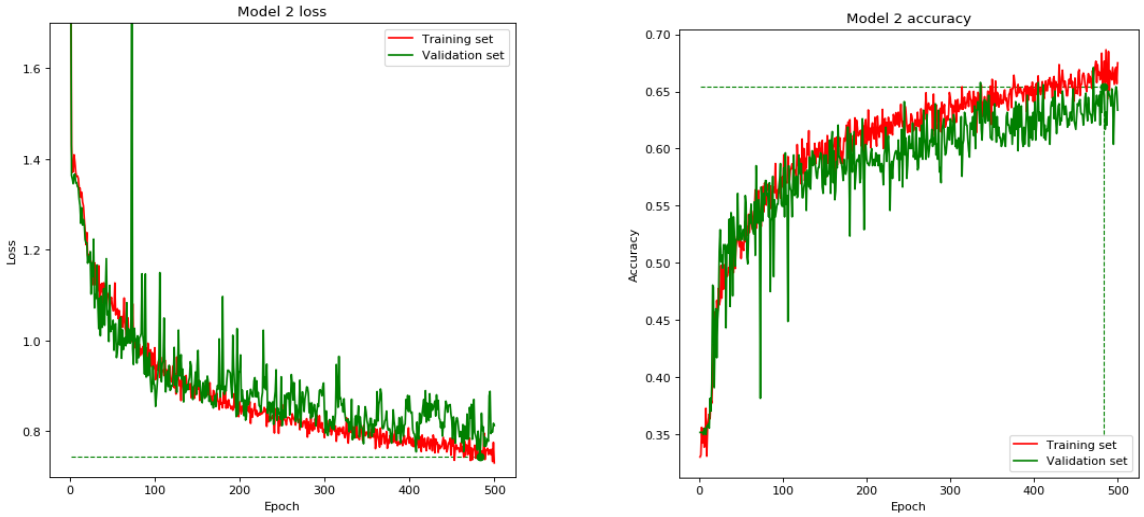
The second model is shallower, but with twice the filters than before in the corresponding layers. Alternatively, it can be seen as a clone of the previous where the first conv block has been skipped. The hypothesis to verify is that more filters in early convolutional stages can help achieving a better accuracy. Specifically:

- Three conv2D layer with 64, 128, 256 filters
- The last FC layer has few more neurons (64, previously was 48) to cope with the increased filters

```
Conv2D(64, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(128, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(256, (3, 3), relu)
MaxPooling2D((2, 2))
Flatten()
Dense(64, relu)
Dropout(0.5)
Dense(4, softmax)
```

The results are shown below.

Experiment 2				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
483	65.42	61.31	0.7440	0.9047



(a) Loss (b) Accuracy

Figure 21: Scratch CNN 2 (4-class)

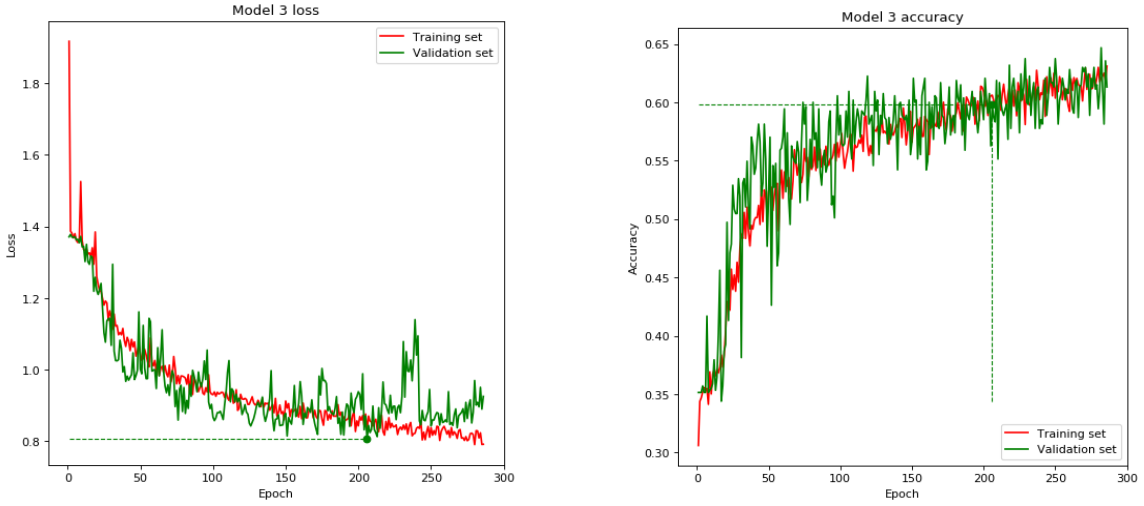
41 samples were assigned the wrong abnormality type, that is a 87.8% accuracy in predicting that feature. On the other side, 30.6% were incorrectly classified as benign or malignant, so the harmfulness prediction accuracy is only 69.4%. The overall accuracy is better than in the previous model (61.3% vs 57.7%). Even after a very long training the loss was still decreasing; increasing the learning rate would be a risky move here, since the evolution is already quite noisy.

4.4.3 Experiment 3 - Deeper

What about another convolutional block? Does it help distinguishing between malignant and benign bodies? The next experiment provides an answer to these question, by training a deeper model with an extra Conv2D layer with 512 features.

```
Conv2D(64, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(128, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(256, (3, 3), relu)
MaxPooling2D((2, 2))
Conv2D(512, (3, 3), relu)
MaxPooling2D((2, 2))
Flatten()
Dense(64, relu)
Dropout(0.5)
Dense(4, softmax)
```

Experiment 3				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
206	59.81	61.01	0.8071	0.9422



(a) Loss (b) Accuracy

Figure 22: Scratch CNN 3 (4-class)

After more than 200 epoch, the NN reached a classification accuracy of 61%. It is interesting to note how the model is more prone to classify malignant abnormalities as benign rather than viceversa. In a medical context it could be a desirable feature, since false negatives are potentially more harmful than false positives.

5 Pre-trained models

This section describes the results obtained on the 2-class and 4-class tasks by means of state-of-the-art models pretrained on other large-scale image datasets.

5.1 VGG16 2-class

VGG16 is a convolutional neural network model proposed by Simonyan et al., with several 3x3 convolutional layers in cascade occasionally interleaved with 2x2 max-pooling layers [21]. Developed for the ILSVRC2014 challenge, it was able to achieve a top-5 accuracy of 92.7% on ImageNet. Its enormous size makes the training an extremely slow process; nevertheless, VGG16 is often used for transfer learning, thanks to its flexibility.

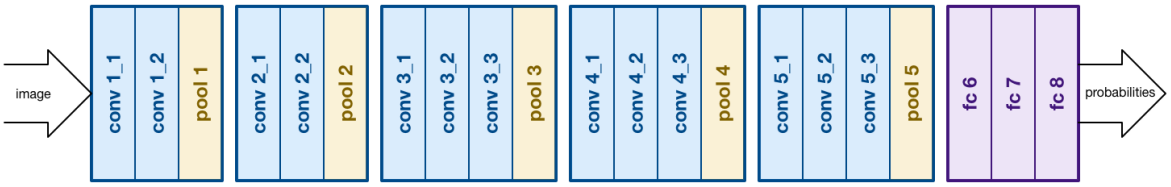


Figure 23: VGG16 Architecture

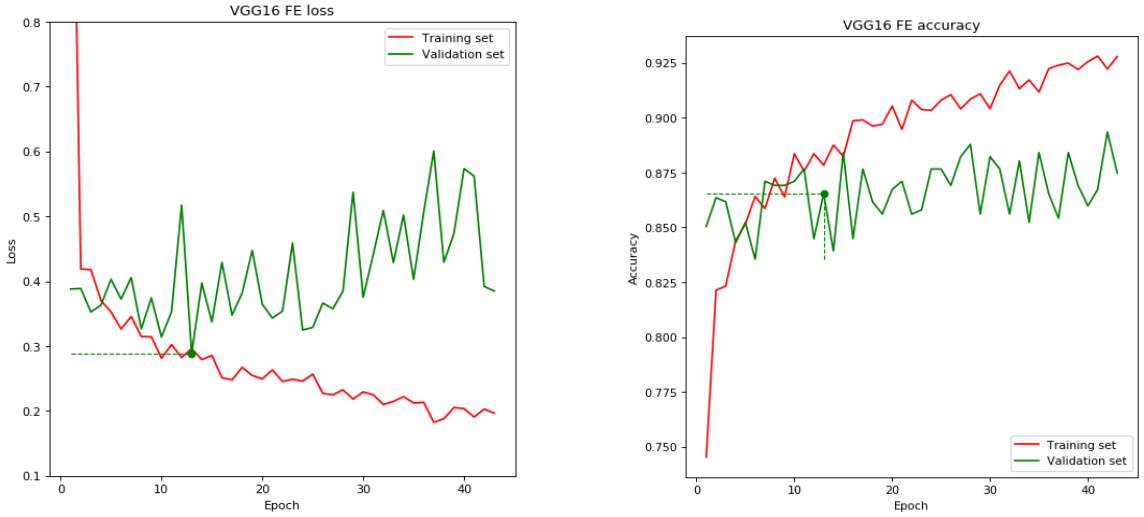
5.1.1 Feature Extraction

Feature extraction is a transfer learning approach which consists in taking a model already trained on another dataset (possibly similar), chopping the final fully-connect layer off and replace it with a new one (i.e. with new weights). The convolutional base is frozen, that is made non-trainable, because it serves as feature extractor for the new images and should not be altered by the training process.

Vanilla
The original VGG16 comes with a couple of 4096 FC layers followed by 1000 softmax neurons, which is alright for ImageNet but definitely oversized for our purpose. Hence, the convolutional base is left as it is, and the fully-connect block is shrinked to a single layer with 256 neurons feeding one sigmoidal output unit.

Feature extraction (vanilla)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
13	86.54	86.90	28.86	43.20

The network learns very fast to distinguish between masses and calcifications, with more than 86% accuracy. On the other hand, by looking at the loss graph, it begins overfitting very soon: some regularization method is very needed.



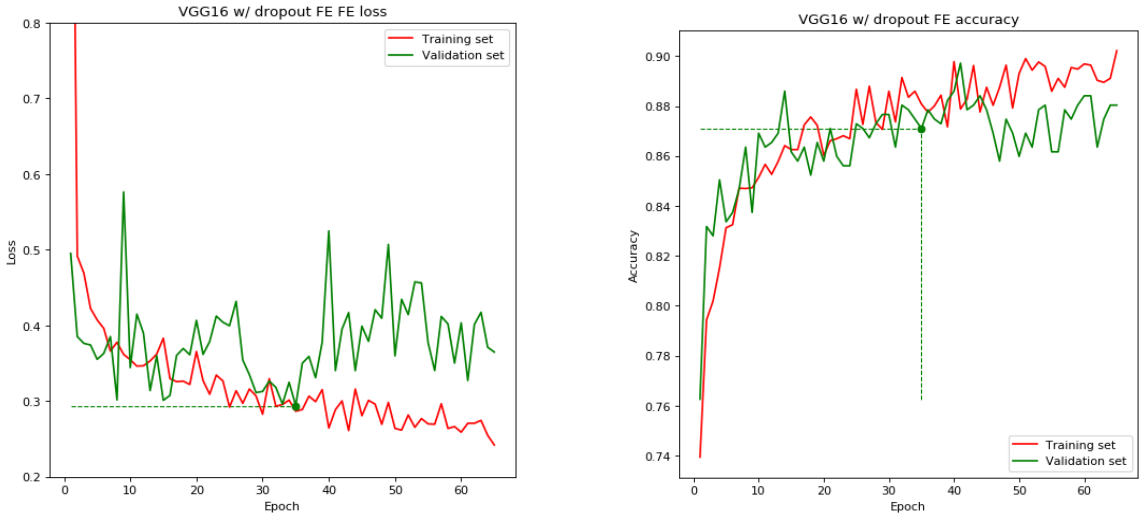
(a) Loss (b) Accuracy

Figure 24: VGG16 Feature Extraction (vanilla)

With dropout

Dropout is added at the end of the network to alleviate overfitting.

Feature Extraction (w/ dropout)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
35	87.10	87.50	0.2940	0.4637



(a) Loss (b) Accuracy

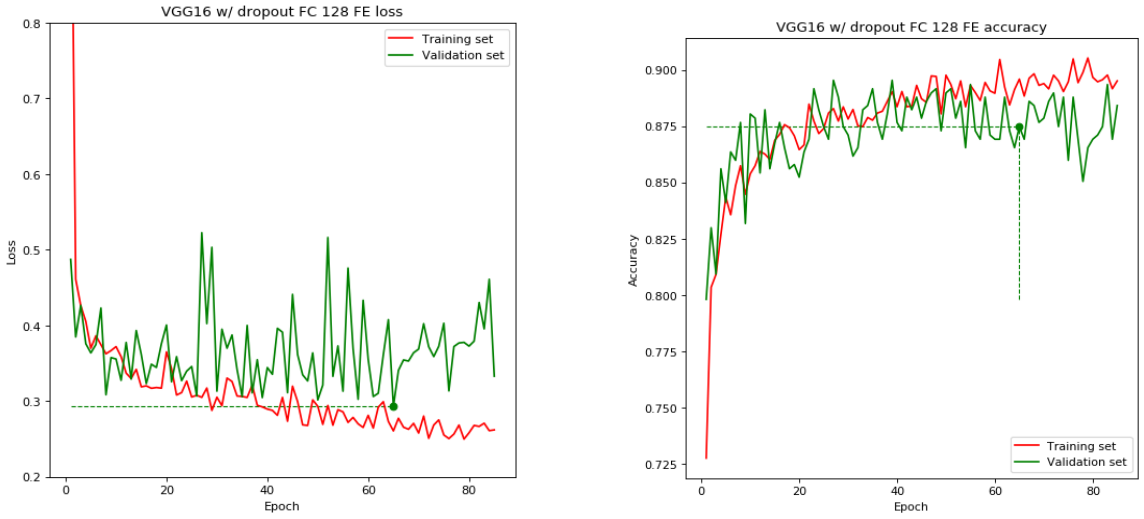
Figure 25: VGG16 Feature Extraction with dropout

As expected, dropout mitigated the magnitude of overfitting, but it was not enough to eliminate it. Overall, the training time increased and the testing accuracy improved to 87.5%.

With dropout and smaller FC

Another way to reduce overfitting is to shrink the network size, e.g. the FC layer. Note that this approach is discouraged when the number of neurons is already relatively low w.r.t. the problem complexity: doing so will likely result in a drop of accuracy. The fully-connected layer now has 128 neurons instead of 256.

Feature Extraction (w/ dropout and smaller FC)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
65	87.48	86.61	0.2930	0.4261



(a) Loss (b) Accuracy

Figure 26: VGG16 Feature Extraction with dropout and smaller FC layer

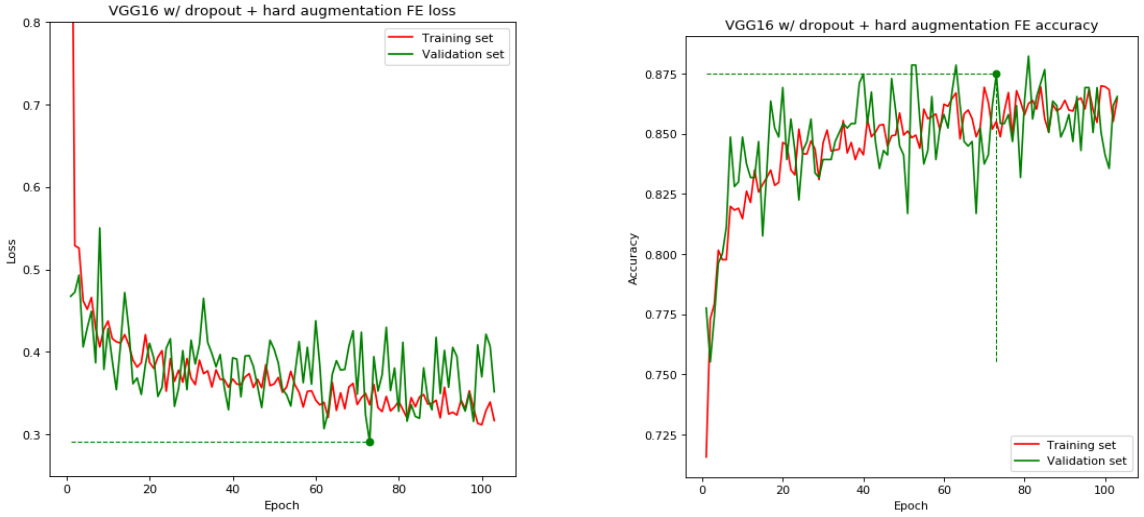
By shrinking the last fully-connected layer and with the use of dropout, overfitting went down even more. However, the FC resize slightly hurted the performance, in terms of training time and final accuracy.

Hard data augmentation

The training set is already augmented, yet its intensity can be increased with more radical transformations. The next experiment extends the shearing range to 15° (from 10°) and introduces shifts up to 20% of the size. The aim is to have a more variegated training set so to reduce overfitting.

Feature Extraction (w/ dropout and more augmentation)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
73	87.48	85.71	0.2910	0.3940

The use of a stronger data augmentation drastically reduced the overfit. The test accuracy remained more or less unchanged, which indicates that the network performance has been already pushed to the limit.



(a) Loss (b) Accuracy

Figure 27: VGG16 Feature Extraction with dropout and stronger data augmentation

5.1.2 Fine tuning

Fine tuning extends feature extraction, unfreezing (i.e. making trainable) one or more of the last convolutional layers after the new FC block has been trained. Training for few additional epochs, possibly with a very low learning rate, these convolutional layers get a change to be slightly adjusted (fine-tuned) to boost performance.

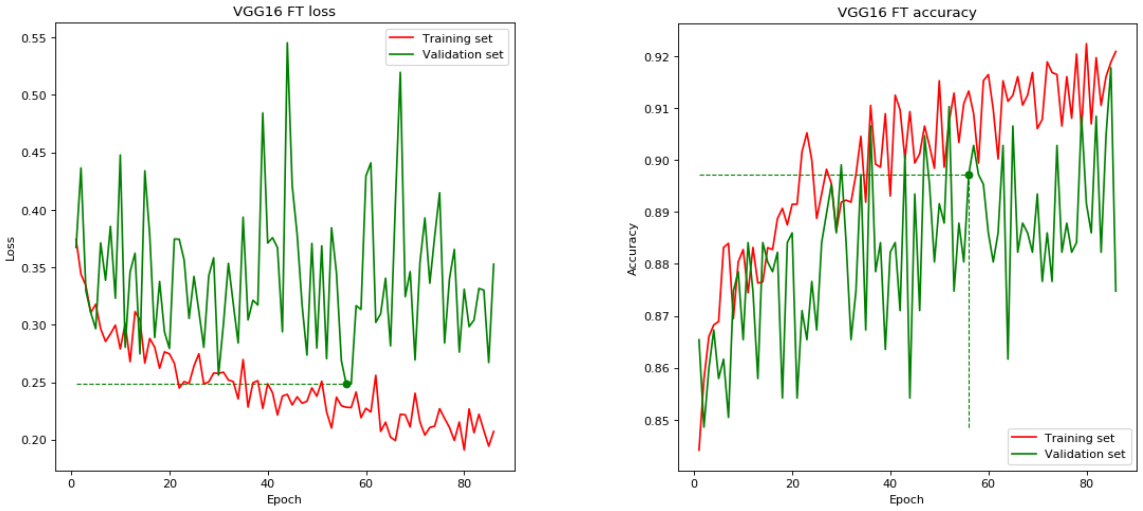
One layer
Starting from the last model (feature extraction with hard augmentation), the 3rd Conv2D layer in the 5th block is unfrozen and the network trained with a learning rate $\mu = 0.0001$, that is 10 times smaller than usual. After a while, a new validation loss minimum is reached, as shown in Fig. 28(a).

Fine tuning (1 layer)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
56	89.72	89.58	0.2484	0.3228

Thanks to fine tuning, the NN scored 89.58% accuracy on the testing set, with a gain of 1% compared to the previous feature extraction VGG16 models.

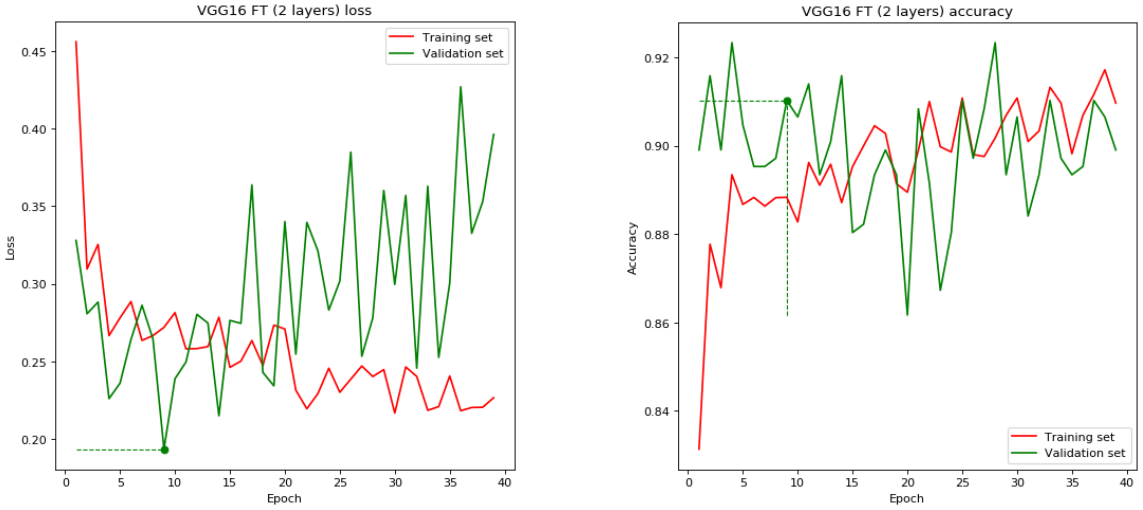
Two layers
The same process can be applied iteratively, this time starting from the 1-layer fine-tuned model and unfreezing the penultimate convolutional layer.

Fine tuning (2 layers)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
9	91.03	91.37	0.4995	0.3312



(a) Loss (b) Accuracy

Figure 28: VGG16 2-class Fine Tuning (1 layer)



(a) Loss (b) Accuracy

Figure 29: VGG16 2-class Fine Tuning (2 layers)

After very few epochs, the validation loss dropped very low and the accuracy reached the outstanding value of 91.4% on the testing set. The network started immediately after; indeed, unfreezing the second layer left too many network parameters free for tuning, so the result we got is basically the best we can get from it. For this reason, fine-tuning a 3rd layer would not be helpful at all.

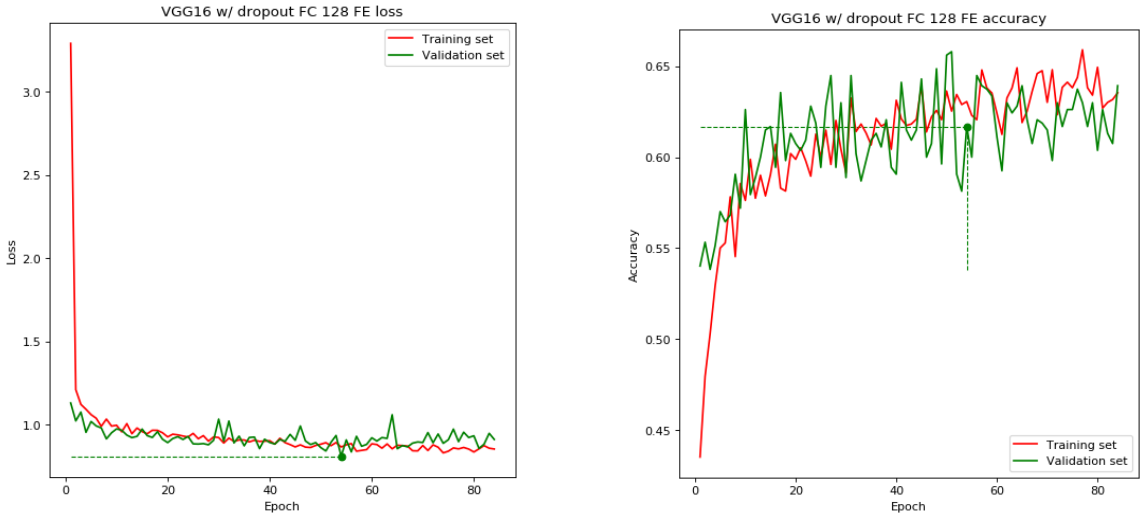
5.2 VGG16 4-class

5.2.1 Feature Extraction

128 FC

Feature extraction with VGG16 on 4 classes is very similar to the 2-class case. The main difference is in the output layer, composed of 4 softmax neurons instead of a single sigmoidal one. The first experiment is run with a fully-connected layer of 128 units. Dropout is enabled with $p = 0.5$ as usual.

Feature Extraction (128 FC)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
54	61.68	56.85	0.8101	0.9742



(a) Loss (b) Accuracy

Figure 30: VGG16 4-class Feature Extraction (128 FC)

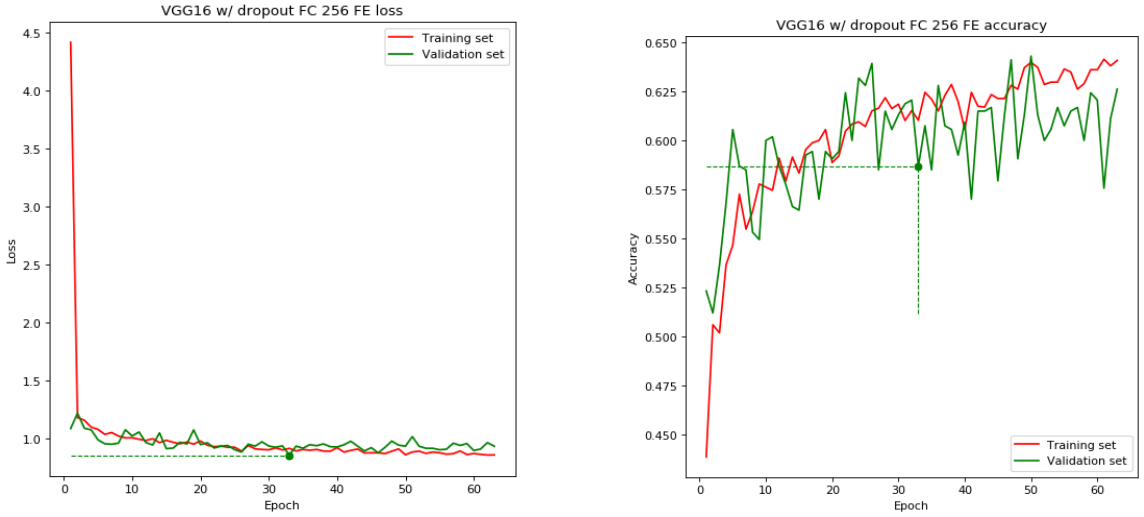
The minimum validation loss was reached after 52 epochs, and is equal to 56.8%. The overfitting here is negligible, and the training loss diminishes very slowly after a while.

256 FC

The same experiment is repeated with a fully-connected layer of 256 elements.

Feature Extraction (256)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
33	58.69	57.74	0.8569	0.9836

Although the final testing accuracy slightly increased, doubling the number of FC parameters did not significantly change the performance but, instead, the network started overfitting earlier. A size of 128 is hence the most reasonable choice.



(a) Loss (b) Accuracy

Figure 31: VGG16 4-class Feature Extraction (256 FC)

5.2.2 Fine tuning

The last convolutional layers of VGG16 can be fine-tuned as done before. With more free parameters, networks naturally tend to overfit, so it is better to start from the FC 128 model rather than 256.

One layer
Last layer is fine-tuned with RMSprop and a learning rate of 1e-4.

Fine Tuning (1 layer)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
16	66.92	56.85	0.7546	0.9586

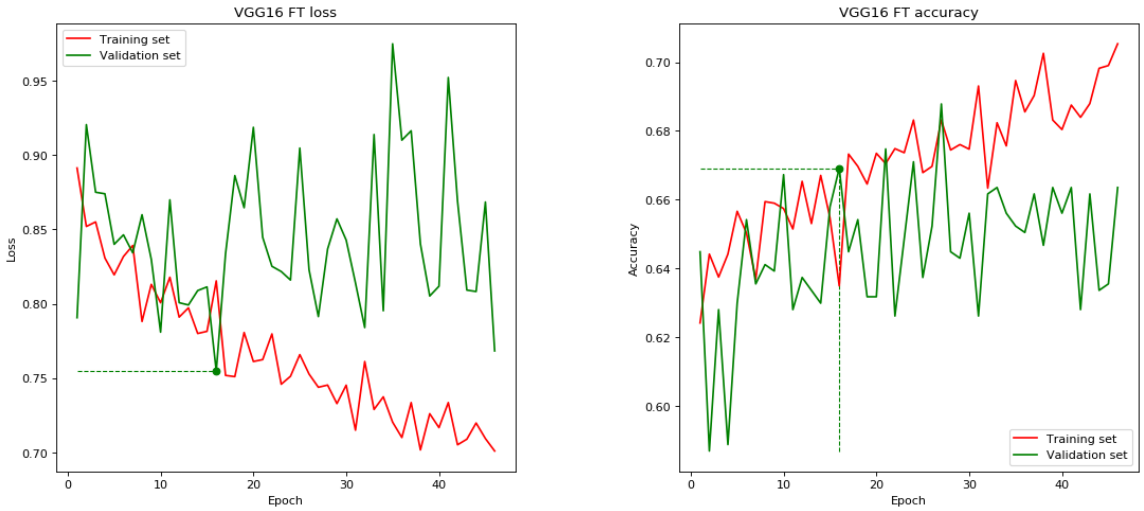
The testing accuracy did not change at all, but the validation loss reached a global minimum w.r.t. the previous experiments.

Two layers
A second convolutional layer can be adjusted starting from optimal model of the previous experiment.

Fine tuning (2 layers)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
21	67.29	60.42	0.7295	1.0290

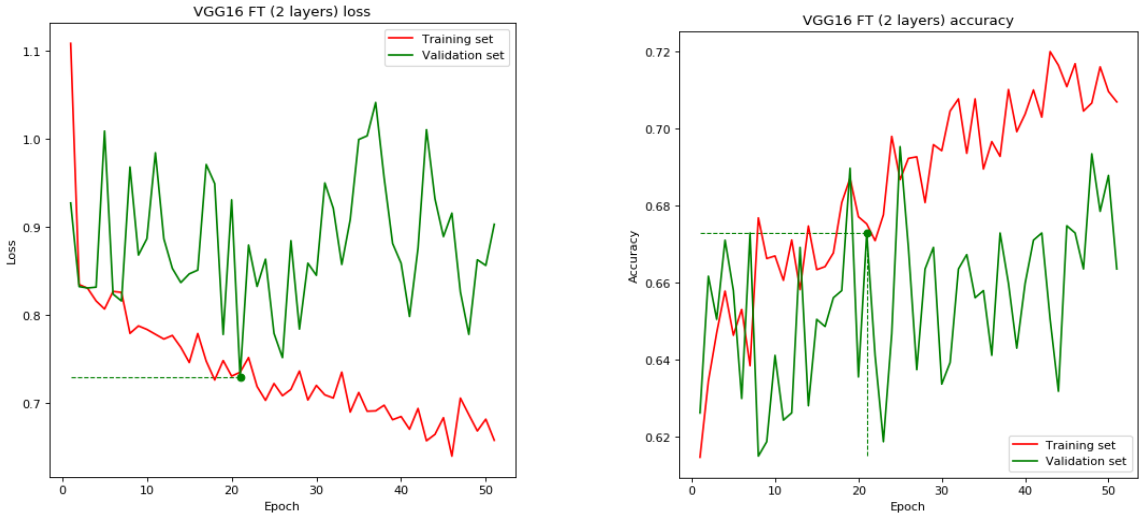
The testing accuracy greatly improved, overcoming the threshold of 60%.

Three layers
A third is eventually tuned with a learning rate equal to 1e-5.



(a) Loss (b) Accuracy

Figure 32: VGG16 4-class Fine Tuning (1 layer)

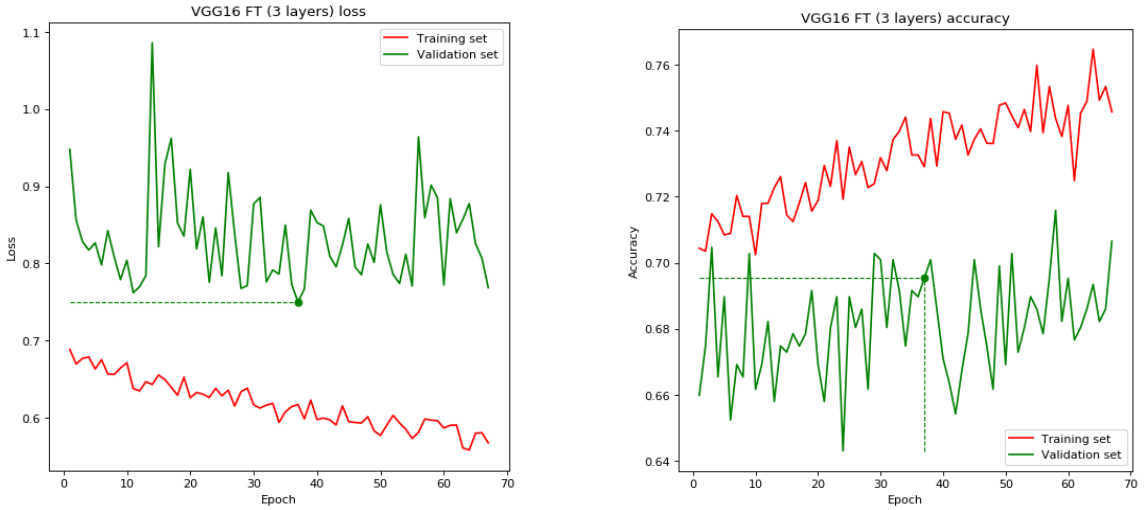


(a) Loss (b) Accuracy

Figure 33: VGG16 4-class Fine Tuning (2 layers)

Fine tuning (3 layers)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
37	69.53	59.23	0.7499	1.0493

The NN became too flexible after unfreezing one more set of parameters, so it failed to achieve a better result than the previous experiment. The accuracy slightly dropped to 59.23% as a consequence.



(a) Loss (b) Accuracy

Figure 34: VGG16 4-class Fine Tuning (3 layers)

5.3 ResNet50 2-class

ResNet, short for Residual Network, is a convolutional neural network with 50 layers developed by He et al. for the ILSVRC 2015 [5], where it secured first place. It is considered one of the best state-of-the-art models, thanks to the disruptive idea of residual layers (bypass some layers to avoid vanishing gradient).

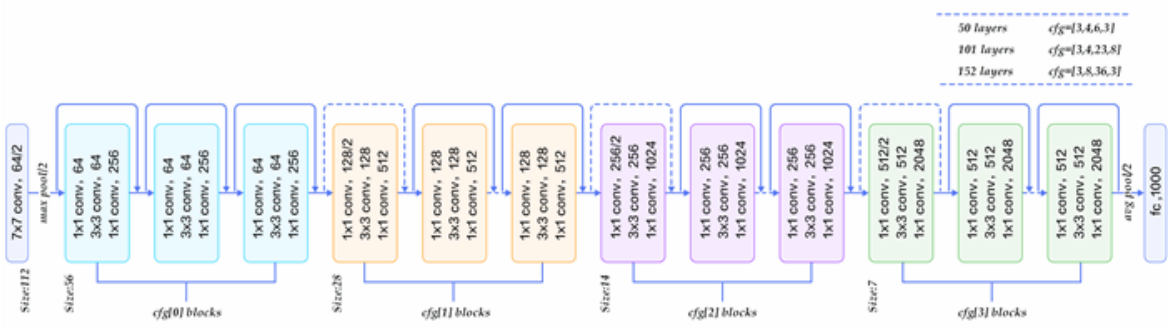


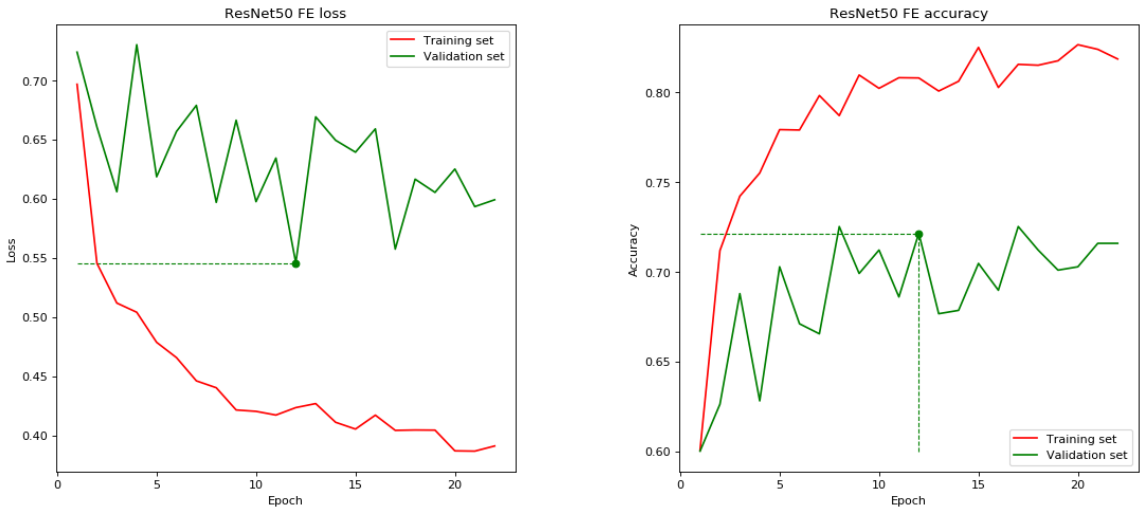
Figure 35: ResNet50 architecture

5.3.1 Feature extraction

ResNet50, being trained on ImageNet, ends with 1000 softmax neurons. In this binary classification task those units are replaced with a single sigmoidal neuron, directly connected to all the 2048 1x1 convolutional feature maps.

Vanilla
A plain ResNet50 with a single final sigmoidal neuron.

Feature Extraction (vanilla)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
12	72.15	74.40	0.5457	0.5948



(a) Loss (b) Accuracy

Figure 36: ResNet50 2-class Feature Extraction (vanilla)

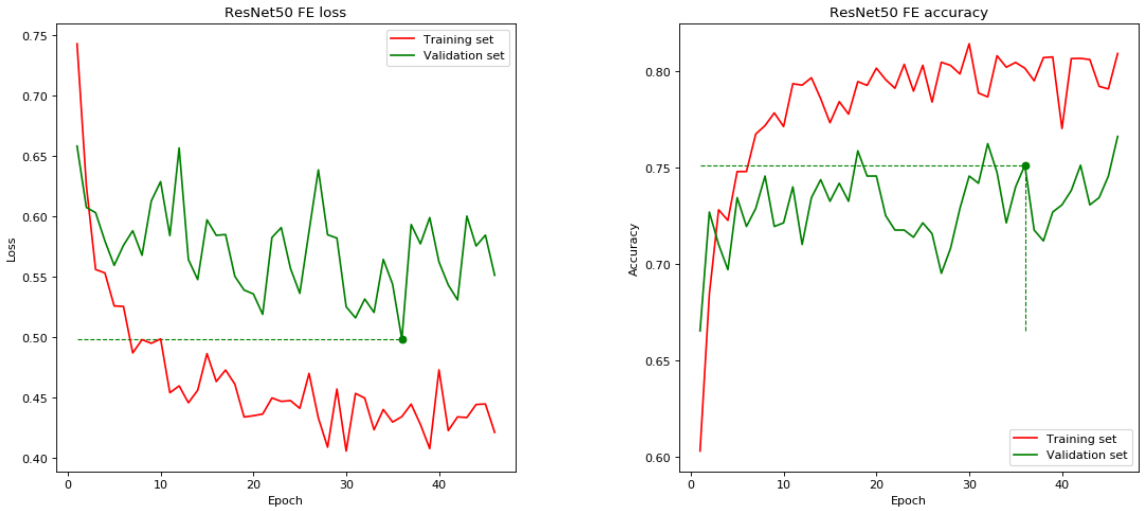
Even with only one neuron in the FC stage, the network starts overfitting after few epochs. The accuracy is low: less than 75%.

With dropout

A dropout layer (p=0.5) has been added before the output layer in an effort to dampen the hard overfitting.

Feature Extraction (w/ dropout)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
36	75.14	76.19	0.4982	0.5428

The addition of a dropout layer reduced overfitting, but the issue is still present and jeopardizes any attempt of training. ResNet50 is perhaps too powerful, and the extracted features are simply too many for a relatively simple task like this. For the sake of comparison, the top accuracy was 76.19%, even lower than the naive model built from scratch. Feature extraction with ResNet50 was a failed experiment.



(a) Loss (b) Accuracy

Figure 37: ResNet50 2-class Feature Extraction (w/ dropout)

5.4 MobileNet

MobileNet is a convolutional neural network architecture proposed by Howard et al. for mobile and embedded vision applications. Its major feature is the use of depth-wise separable convolutions, and hyperparameters that can be adjusted to trade off between latency and accuracy [6].

5.4.1 Feature extraction

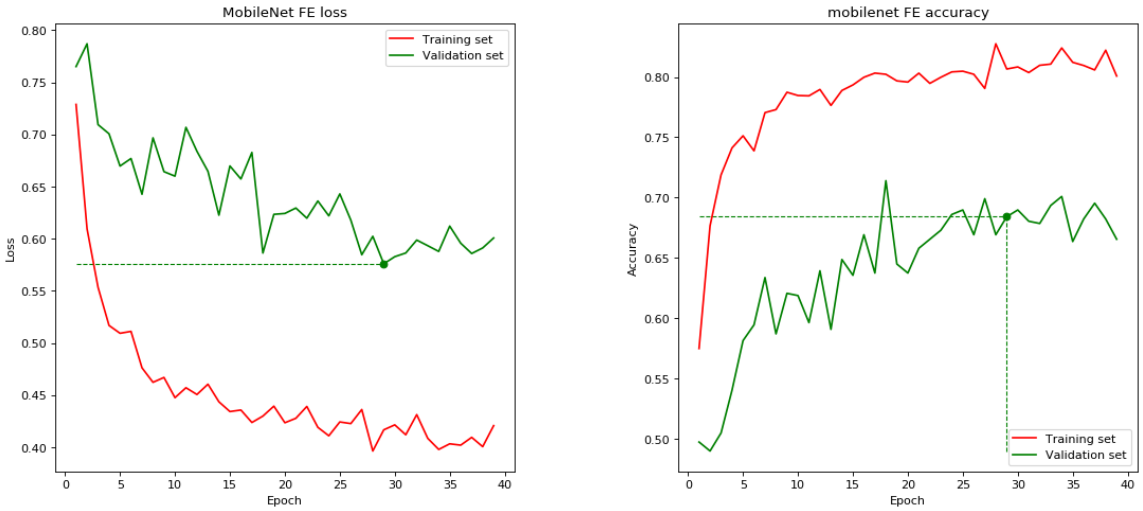
In Keras, MobileNet is available with pretrained weights in different flavors. The input size can be adjusted (160x160 is the closest to our dataset) as well as the number of filters (α=0.5 here).

Vanilla
A single output neuron is fully-connected to the convolutional base, which is the bare minimum to have a functional network. If the network underfits, then more neurons will be added later.

Feature Extraction (vanilla)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
29	68.41	70.24	0.5759	0.6054

The validation loss is constantly very distant from the training one (about 0.20 points), so does the validation accuracy which fails to get above 70%. This result is not promising, and resembles the behavior of ResNet50.

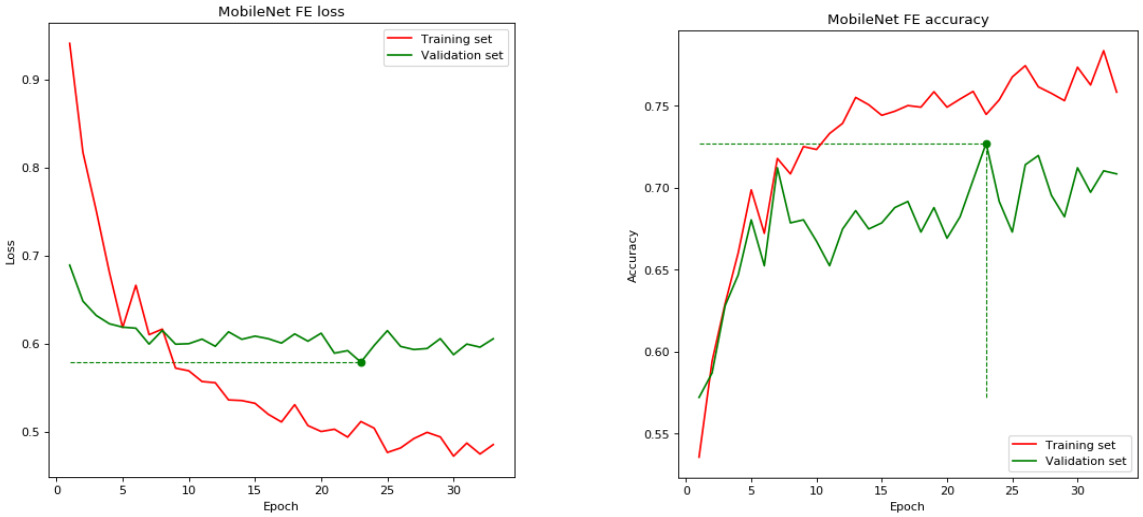
With dropout
A dropout layer (p=0.5) has been added before the output layer to reduce the large gap between the validation and training losses.



(a) Loss (b) Accuracy

Figure 38: MobileNet 2-class Feature Extraction (vanilla)

Feature Extraction (w/ dropout)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
23	72.71	76.19	0.5788	0.5858



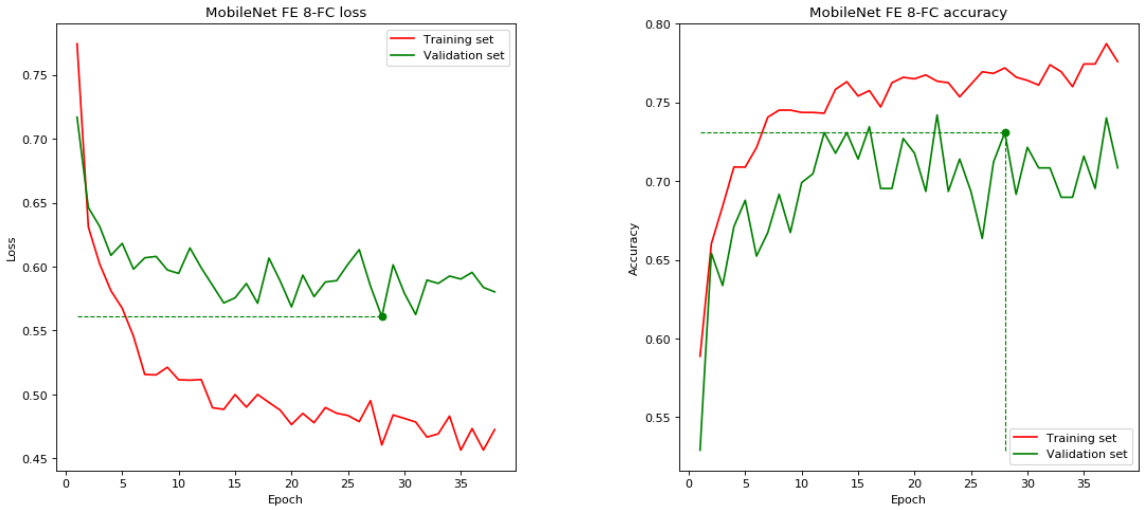
(a) Loss (b) Accuracy

Figure 39: MobileNet 2-class Feature Extraction (w/ dropout)

Overall the network reached a better testing accuracy than before (76.19%). This value is still very low, but at least the loss gap has halved. The training accuracy settles around 76% without growing further, which is a hint of a bottlenecked network (clearly in the FC layer).

Larger FC
In this experiment the fully-connected block is extended with 8 more neurons.

Feature Extraction (FC 8)				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
28	73.08	72.62	0.5608	0.5980



(a) Loss (b) Accuracy

Figure 40: MobileNet 2-class Feature Extraction (FC 8)

The addition of a fully-connected layer resulted in an improvement on the validation set, but not on the testing one. Overall, like Resnet, MobileNet does not work very well as feature extractor on mammogram images.

6 Composite

The 4-class task can be decomposed in two subtasks: mass-calcification classification and benign-malignant diagnosing. The idea is to train two models independently, each focusing on one subtask only; the two networks will then work in parallel, each producing one half of the full 4-class prediction. The rationale behind this *divide et impera* approach is that each network specializes itself to solve a smaller and potentially easier subproblem, thus getting better results than addressing the whole problem in one fell swoop.
The model for mass-calcification classification is the fine-tuned VGG16 network of Section 5.1.2, with a 91.37% accuracy on the test set.
The model for benign-malignant classification is instead a CNN with 3 convolutional blocks (64, 128, 256 channels respectively), 64 fully-connected and dropout 0.5. Trained for more than 400 epochs, it has an accuracy of 69% at distinguishing benign and malignant abnormalities on the test set.
The final prediction is simply the concatenation of the two networks responses. For instance, if the first classifier outputs 'mass' and the second says 'benign', the final label will be 'benign mass'.
In practice, adopting this strategy results in a 62.5% accuracy on the test set, which represents an improvement over the all-in-one 4-class models (the best was 61.3% as described in Sec. 4.4.2).

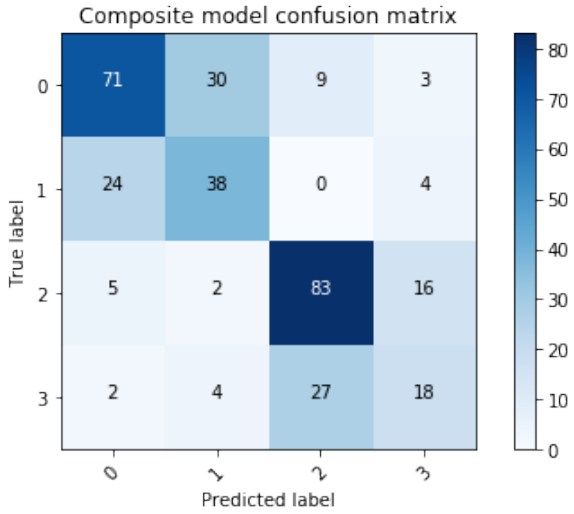


Figure 41: Composite 4-class classifier confusion matrix

7 Baseline

The dataset contains not only images of masses and calcifications, but also patches of healthy tissue adjacent to the abnormality. The latter show no evidence of disease, however it can be assumed that they still contain information about the neighboring abnormality. Even if this hypothesis turns out to be false, that is no hint of mass/calcification is present, such baseline images can be exploited anyway to normalize the abnormality patches by subtracting irrelevant features.

7.1 Dual channel CNN

The next experiment involves a CNN architecture with two input channels and two separate convolutional bases, joined at their ends by means of a fully-connected layer (Fig. 43. The basic idea is to run two CNNs in parallel, one fed with abnormality patches and the other with baseline ones, then combine the information extracted by both to get a better prediction. The branch trained on abnormalities will do the heavy work, while the baseline one is supposed to help when the classification is ambiguous. The two convolutional branches are trained at the same time with pairs of abnormality-baseline images, but their parameters evolve independently. The only shared weights are in the FC layer.

Dual channel CNN				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
188	90.62	87.20	0.2320	0.4116

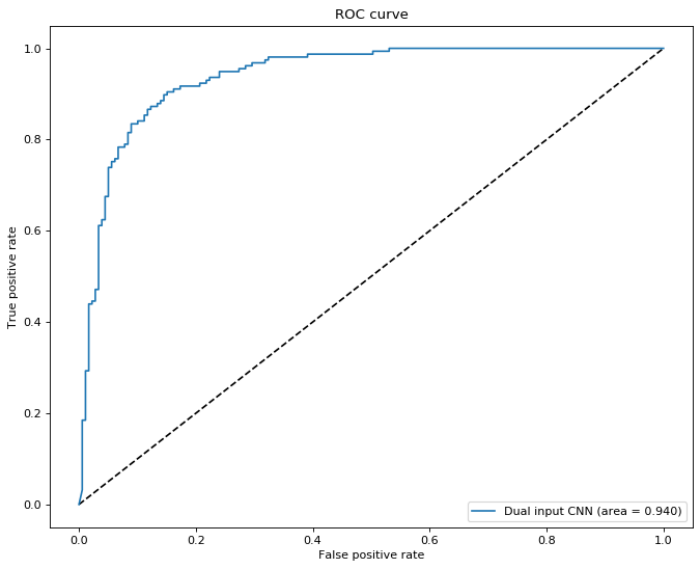


Figure 42: Dual channel CNN ROC

After training for 188 epochs, the network reached an accuracy of 87.2%, which is high but not better than the corresponding single-channel models trained on the

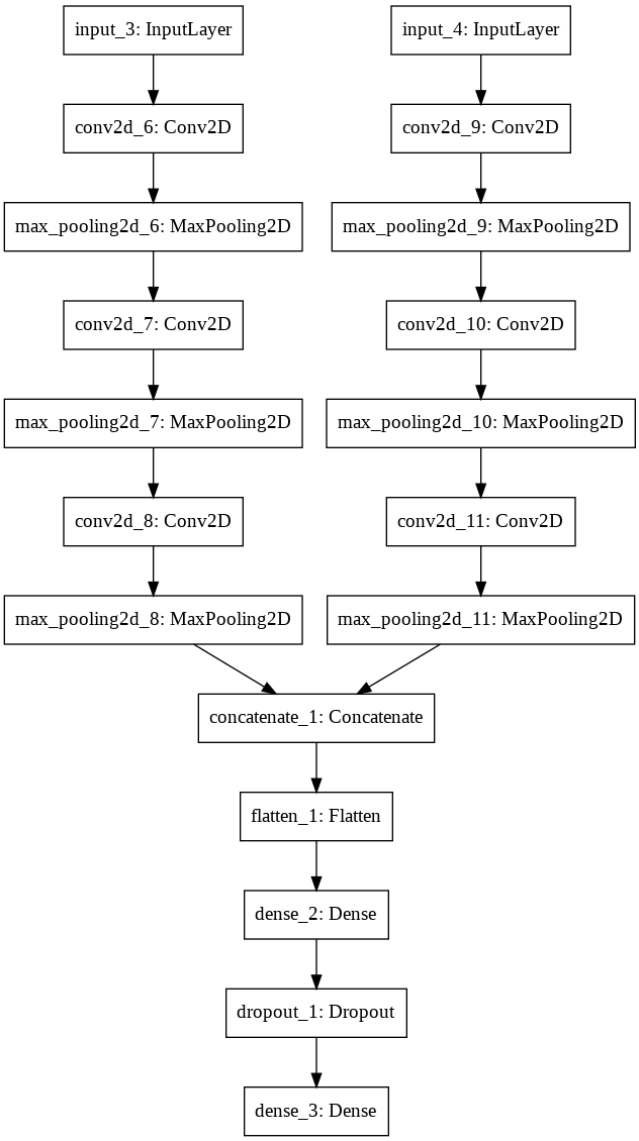


Figure 43: Dual channel CNN

abnormality patches only. This results proves that baseline pictures contain little to no information about the adjacent abnormality type.

7.2 Siamese CNN

A Siamese CNN uses the same weights to extract features from two distinct inputs concurrently, then it computes their difference according to a predefined distance metric. Such an unconventional architecture was first proposed by Bromley et al. for signature verification, and found later applications in the domains of one-shot learning and object tracking [2]. Here the the baseline patches are exploited to extract features that do not depend from the abnormality, but only on specific local characteristics of the patient. The same features are likely to be found in the abnormality patch of the same patient, as there is a natural correlation between adjacent tissues of the same person. Cancelling these features (by subtraction) is supposedly a way to isolate and enhance those features that instead originate from masses/calcifications, ultimately improving the classification reliability.

Siamese CNN				
Epoch stop	**Validation accuracy**	**Testing accuracy**	**Validation loss**	**Testing loss**
446	90.43	86.61	0.2561	0.3530

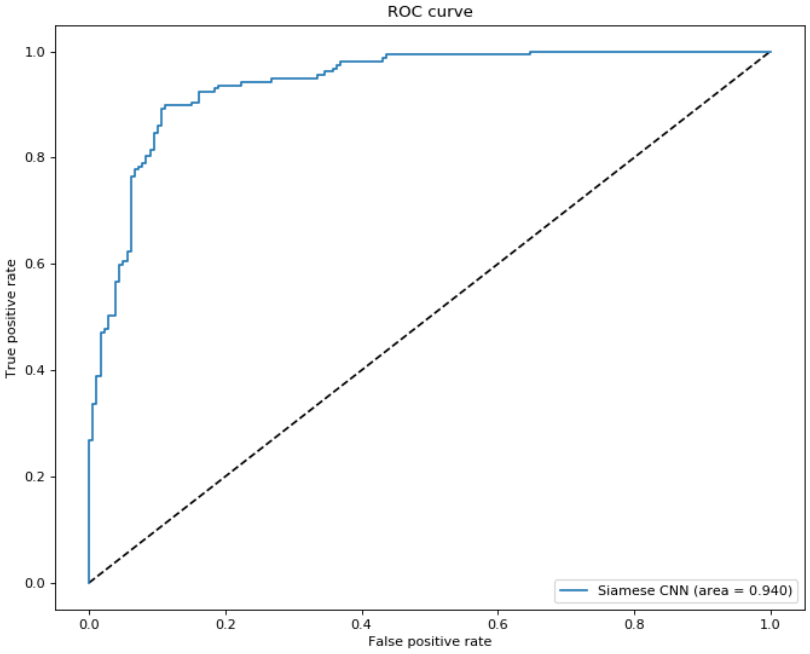


Figure 44: Siamese CNN ROC

The Siamese network has an accuracy of 86.61% on the testing set. Although slightly below the score of other models, the result is very interesting because it was achieved with a radically different approach.
Note also that Dual Channel and Siamese obtained the same AUC score: 0.94.
When multiple diverse models achieve similar results independently, it can be interpreted as a proof of soundness for the proposed approach.

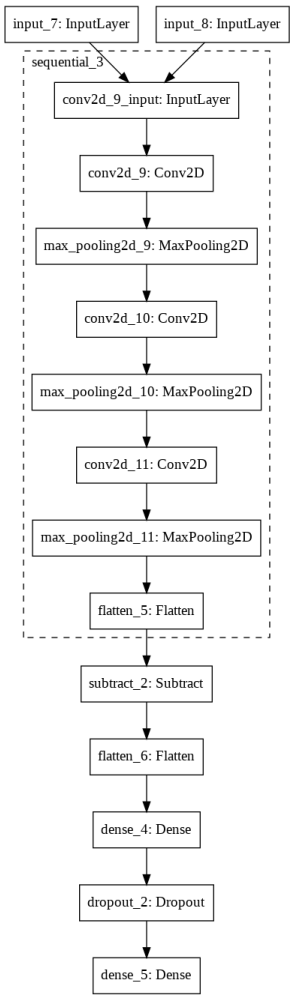
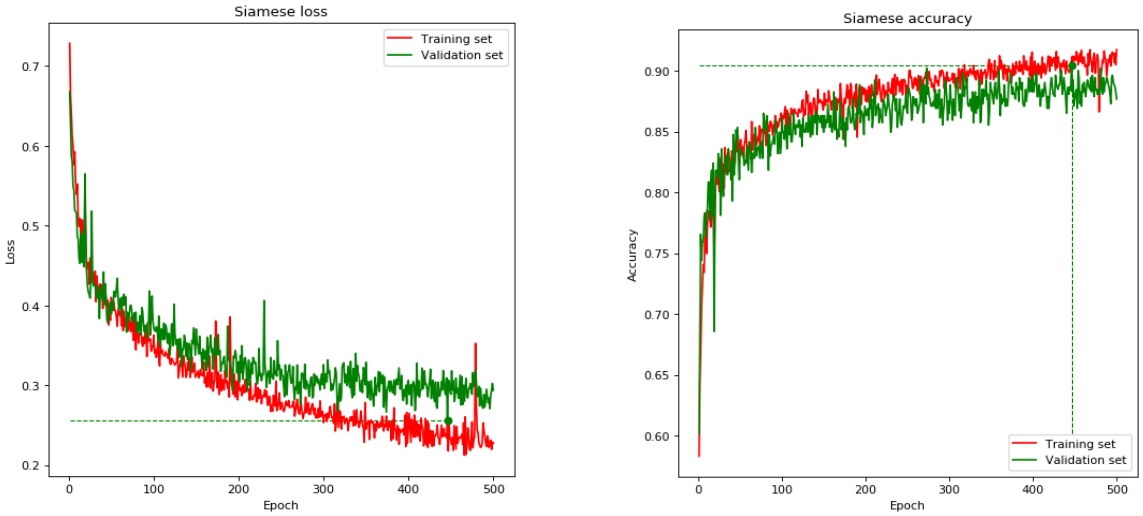


Figure 45: Siamese CNN



(a) Loss (b) Accuracy

Figure 46: Siamese CNN

8 Ensemble

An *ensemble* is basically a collection of neural network models trained to solve a specific task, which are combined to obtain a performance that is often superior to those of the individual models. The idea is to build a strong classifier out of several weaker models. There are many ways to merge the predictions of the different networks into a single one. Typical schemes are voting, averaging and weighted averaging, but other strategies are possible too. Voting means choosing the most common option (mode), while averaging computes the mean of the individual predicted values. Technically, even an ensemble whose components have the same architecture but different weights can result in marginal improvements; nevertheless, diversity is encouraged as it maximizes the effectiveness of this approach.

8.1 2-class

8.1.1 3-ensemble

Three classifiers are chosen among the 'networks from scratch' of Section 4. Models 5, 6 and 9 have slightly different architectures but similar test accuracy.

Model	Weight	**Test accuracy**
Model 5	1	88.69
Model 6	1	87.80
Model 9	1	88.10
Voting		89.58
Averaging		88.99

Table 1: 3-ensemble for mass-calcification classification

The ensemble with a voting strategy (mode) to produce the final prediction has a 89.58% accuracy, which is 0.89% higher than the best individual model. Averaging, instead, leads to a 88.99% test accuracy, with a minimal improvement of 0.30%.

8.1.2 4-ensemble

Another ensemble is constructed by including also the fine-tuned VGG16 model in the previous ensemble. Since the VGG network is more reliable than the other individuals by few points of accuracy, it is given a weight of 2, that is equivalent to considering that model twice.
Interestingly, the ensemble with the VGG network did not outperform the VGG itself. The other three models did not help VGG at all, but instead made confusion on some samples which eventually resulted in an accuracy drop of 0.8%. That is, VGG16 alone does perform better than the ensemble crafted around it.

8.2 4-class

This ensemble is composed of the three 4-class models developed from scratch.

Model	Weight	**Test accuracy**
Model 5	1	88.69
Model 6	1	87.80
Model 9	1	88.10
VGG16 FT	2	91.87
Voting		91.07
Averaging		91.07

Table 2: 4-ensemble for mass-calcification classification

8.2.1 3-ensemble

Model	Weight	**Test accuracy**
Model 1	1	57.74
Model 2	1	61.31
Model 3	1	61.01
Voting		59.82
Averaging		61.31

Table 3: 3-ensemble for four-class classification

Averaging did not improve the accuracy at all, while voting made it even worse with respect to the best individual model. This demonstrates that ensembles often lead to better results, but they are not guaranteed to do so; in some unfortunate cases, like this, an ensemble can perform worse.

8.2.2 4-ensemble

The 4-class fine-tuned VGG model is added to the previous ensemble. Since the number of classifiers is now even, voting would not be predictable; averaging is the most natural choice here.

Model	Weight	**Test accuracy**
Model 1	1	57.74
Model 2	1	61.31
Model 3	1	61.01
VGG16 FT	1	59.23
Averaging		61.31

Table 4: 4-ensemble for four-class classification

The addition of another model didn't change the classification accuracy, which is stuck at 61.3%.

8.2.3 Weighted ensemble with grid search

The previous averaging ensemble gave the same importance to all the four models, although this is not always the best option. A good ensemble should encourage diversity, that is models whose predictions are positively correlated for the correct samples and negatively correlated for the mistakes. By assigning different weights to each model, the final ensemble may give better results; the challenge is indeed to find the best combination of such weights. A simple approach in this sense is *grid search*, which is basically an exhaustive search through a subset of all the possible configurations. Hold-out can be used have a validation set where to measure the goodness of the weights, and a testing set to evaluate the actual accuracy.
For this dataset, since the test set is already small, splitting it into two subsets (validation and testing) could easily end up in distribution asymmetries that would nullify any result from a statistical point of view. To avoid this situation, the average accuracy obtained on validation and test set is monitored: if they differ too much, it means that one subset contains more hard samples than the other, and the experiment should be repeated with a different partitioning.
In this case the testing set is split in half. Grid search is performed on the 4-ensemble with weights ranging from 0 to 1 in steps of 0.1. The combination that gets the best accuracy on the validation set is selected as optimal and evaluated on the test set. Since the predictions of each network for each sample can be precomputed (and scaled later for averaging), the overall search is relatively fast.

Model	Weight	**Test accuracy**
Model 1	0.2	57.74
Model 2	0.1	61.31
Model 3	0.2	61.01
VGG16 FT	0.5	59.23
Averaging		62.5

Table 5: Weighted 4-ensemble for four-class classification with grid search

The optimization procedure returned (0.2, 0.1, 0.2, 0.5) as the optimal combinations of weights to associate to the ensemble models. With an average accuracy of 61.8% on the validation set and 61.1% on the test one, the partitioning appears to be fair, not excessively biased towards any subset. These weights score a 62.5% accuracy on the new test set, which is a good improvement w.r.t. individual models and the previous ensembles.

References

[1] Richa Agarwal et al. “Mass detection in mammograms using pre-trained deep learning models”. In: *14th International Workshop on Breast Imaging (IWBI 2018)*. Vol. 10718. International Society for Optics and Photonics. 2018, 107181F.

[2] Jane Bromley et al. “Signature verification using a” siamese” time delay neural network”. In: *Advances in neural information processing systems*. 1994, pp. 737–744.

[3] Arthur C Costa et al. “Data augmentation for detection of architectural distortion in digital mammography using deep learning approach”. In: *arXiv preprint arXiv:1807.03167* (2018).

[4] Neeraj Dhungel, Gustavo Carneiro, and Andrew P Bradley. “Automated mass detection in mammograms using cascaded deep learning and random forests”. In: *2015 international conference on digital image computing: techniques and applications (DICTA)*. IEEE. 2015, pp. 1–8.

[5] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[6] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[7] Zeshan Hussain et al. “Differential data augmentation techniques for medical imaging classification tasks”. In: *AMIA Annual Symposium Proceedings*. Vol. 2017. American Medical Informatics Association. 2017, p. 979.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[9] Rebecca Sawyer Lee et al. “A curated mammography data set for use in computer-aided detection and diagnosis research”. In: *Scientific data* 4 (2017), p. 170177.

[10] Daniel Lévy and Arzav Jain. “Breast mass classification from mammograms using deep convolutional neural networks”. In: *arXiv preprint arXiv:1612.00542* (2016).

[11] Bruno Roberto Nepomuceno Matheus and Homero Schiabel. “Online mammographic images database for development and comparison of CAD schemes”. In: *Journal of digital imaging* 24.3 (2011), pp. 500–506.

[12] Scott Mayer McKinney et al. “International evaluation of an AI system for breast cancer screening”. In: *Nature* 577.7788 (2020), pp. 89–94.

[13] Dmytro Mishkin. *Batch normalization before or after Relu*. `https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md`. 2016.

[14] Inês C Moreira et al. “Inbreast: toward a full-field digital mammographic database”. In: *Academic radiology* 19.2 (2012), pp. 236–248.

[15] Sungheon Park and Nojun Kwak. “Analysis on the dropout effect in convolutional neural networks”. In: *Asian Conference on Computer Vision*. Springer. 2016, pp. 189–204.

[16] Lutz Prechelt. “Early stopping-but when?” In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[17] Dina A Ragab et al. “Breast cancer detection using deep convolutional neural networks and support vector machines”. In: *PeerJ* 7 (2019), e6201.

[18] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[19] Maha Sharkas, Mohamed Al-Sharkawy, and Dina Ahmed Ragab. “Detection of microcalcifications in mammograms using support vector machine”. In: *2011 UKSim 5th European Symposium on Computer Modeling and Simulation*. IEEE. 2011, pp. 179–184.

[20] Li Shen et al. “Deep learning to improve breast cancer detection on screening mammography”. In: *Scientific reports* 9.1 (2019), pp. 1–12.

[21] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[22] Leslie N Smith. “Cyclical learning rates for training neural networks”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 464–472.

[23] Samuel L Smith et al. “Don’t decay the learning rate, increase the batch size”. In: *arXiv preprint arXiv:1711.00489* (2017).

[24] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[25] P Suckling J. “The mammographic image analysis society digital mammogram database”. In: *Digital Mammo* (1994), pp. 375–386.

[26] Christian Szegedy et al. “Going deeper with convolutions”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[27] Pengcheng Xi, Chang Shu, and Rafik Goubran. “Abnormality detection in mammography using deep convolutional neural networks”. In: *2018 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*. IEEE. 2018, pp. 1–6.