Classification of B-Cell Lymphoblasts versus Normal Cells in White Blood Cells Microscopic Images

Federica Baldi, Daniele Cioffo
MSc in Artificial Intelligence and Data Engineering

TABLE OF CONTENTS

Classification of B-Cell Lymphoblasts versus Normal Cells in White Blood Cells Microscopic Images

Federica Baldi, Daniele Cioffo
MSc in Artificial Intelligence and Data Engineering

1. INTRODUCTION

Acute Lymphoblastic Leukemia (ALL) is a fast-growing cancer of the blood and bone marrow, characterized by the development of many immature lymphocytes called lymphoblasts. It is the most common type of leukemia in children, particularly those between the ages of two and five, and the most common cause of death from pediatric cancers [1].

Treatment depends on the type of ALL, age at diagnosis, and other factors. In any case, early detection and diagnosis are key to a good change for a cure. In order to reach a decisive conclusion for the diagnosis of the disease and the degree of progression, it is of paramount importance to identify malignant cells with high accuracy. The task of distinguishing lymphoblasts from normal white blood cells under the microscope is however generally challenging, as morphologically the images of the two cell types appear similar. Computer-aided tools can be extremely useful in automating the process of identifying malignant cells and in helping pathologist and oncologist make quicker and data-driven inferences.

With this in mind, the goal of our project, inspired by the *ISBI 2019 C-NMC Challenge* [2], is to develop a deep learning architecture that can solve the classification problem in cancer cell imaging.

1.1. Dataset

The dataset that was used in the challenge, named *C-NMC 2019 Dataset* [3], was prepared at Laboratory Oncology, AIIMS, New Delhi, India. It consists of 15114 images of white blood cells with labels (normal versus cancer) that were segmented from microscopic images. The ground truth was marked by the expert oncologist.

The individual microscopic images are three-channel images of size 450×450. They are representative of images in the real-world because they contain some staining noise and illumination errors, although most of these errors have been fixed using stain color normalization [4] [5] [6].

Images were collected from the data of 69 cancer subjects and 49 healthy subjects and were already split into training set, validation set, and test set, as shown in the table below.

	Cancer Cells	**Normal Cells**	***Total***
Training Set	7272	3389	10661
Validation Set	1219	648	1867
Test Set	Unknown	Unknown	2586
Total	Unknown	Unknown	**15114**

With respect to the ground truth labels of the test set, these were not provided. Classification results could be evaluated on the test set by submitting the model to the official competition website and checking the leaderboard. The evaluation metric was the weighted F1-score.

In our case, however, due to the limitations of Google Colaboratory on the use of the GPU, it was not possible to exploit the entire dataset because the use of the CPU alone would have led to too long training times. In order to solve this problem, we decided to use only the training set as the entire dataset and split it in turn into training set, validation set, and test set. The percentages chosen were 60% for the training set, 20% for the validation set, and 20% for the test set. The images to be included in each of the sets were chosen randomly trying to maintain the data distribution of the original dataset. The results are shown in the table below.

	Cancer Cells	**Normal Cells**	***Total***
Training Set	4432	1964	6396
Validation Set	1403	729	2132
Test Set	1437	696	2133
Total	7272	3389	**10661**

2. RELATED WORKS

In recent years, the advent of convolutional neural networks has enabled to achieve excellent results in many image classification problems, including those in the field of medical imaging. With regard to the classification of leukemic B-lymphoblast cells, however, the task is challenging even when leveraging such technologies.

Indeed, the visual similarity between abnormal and normal cells on the one hand, and the great diversity between cells from different subjects on the other hand, makes it particularly difficult to distinguish the two. For this reason, the ISBI Challenge 2019 was presented and, even after its completion, many researchers continue to study the problem.

E. Verma & V. Singh (2019) "*ISBI Challenge 2019: Convolution Neural Networks for B-ALL Cell Classification*" [7] propose an ensemble solution that combines variants of classifiers designed with pretrained MobileNetV2 architecture. Their model achieves a **weighted F1-score** of **0.8947** on the final test set[1].

A. Honnalgere & G. Nayak (2019) "*Classification of Normal Versus Malignant Cells in B-ALL White Blood Cancer Microscopic Images*" [8] address the problem by finetuning a VGG16 network with batch normalization and applying CLAHE to the input images before feeding them to the network. Their model achieves a **weighted F1-score** of **0.8080** on the final test set.

C. Marzahl, M. Aubreville, J. Voigt & A. Maier (2019) "*Classification of Leukemic B-Lymphoblast Cells from Blood Smear Microscopic Images with an Attention-Based Deep Learning Method and Advanced Augmentation Techniques*" [9] propose a solution based upon advanced augmentation techniques and transfer learning. Additionally, they incorporate a basic attention mechanism based on a region proposal subnetwork. Their ensemble of ResNet18 networks achieves a **weighted F1-score** of **0.8284** on the final test set.

F. Xiao, R. Kuang, Z. Ou & B Xiong (2019) "*DeepMEN: Multi-model Ensemble Network for B-Lymphoblast Cell Classification*" [10] propose Deep Multi-model Ensemble Network (DeepMEN), an ensemble of six deep learning models. Their model obtains a **weighted F1-score** of **0.8856** on the final test set.

[1] Please note that the final test set is the one that was used to determine the winners of the challenge and whose labels are not available to us. Our results will therefore be comparable to the state of the art only to a certain extent.

3. METHODS AND EXPERIMENTS

3.1. Data Preprocessing

The training set, as has already been shown, is unbalanced since we have that about 69% of the images belong to the cancer class, while the remaining 31% belong to the normal class. In order to solve this issue, a combination of subsampling and data augmentation techniques were applied so that the number of images of normal cells and of the ones of cancer cells were equalized to 3198.

In particular, from the 4432 cancer cell images, 3198 images were randomly selected. Furthermore, 1234 normal cell images were randomly selected in order to perform data augmentation on them. Noting that microscopic images are invariant to rotations and flips, new images were obtained from the originals by randomly performing a vertical or horizontal flip, or by applying a rotation of a random angle between 90°, 180°, and 270°, as can be seen in Figure 1.

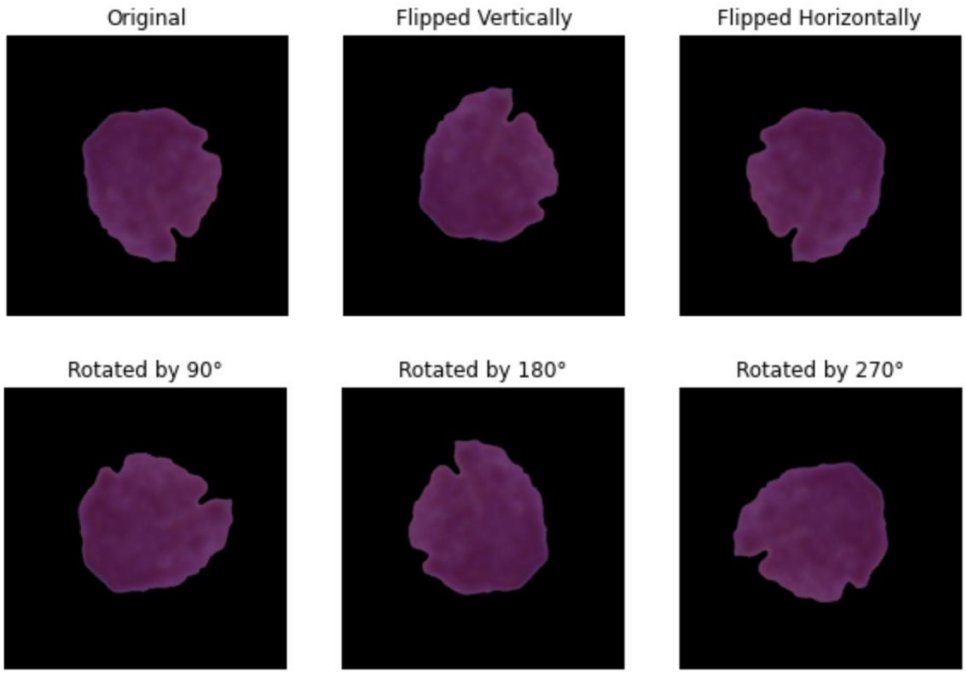


Figure 1. Transformations performed on images for data augmentation

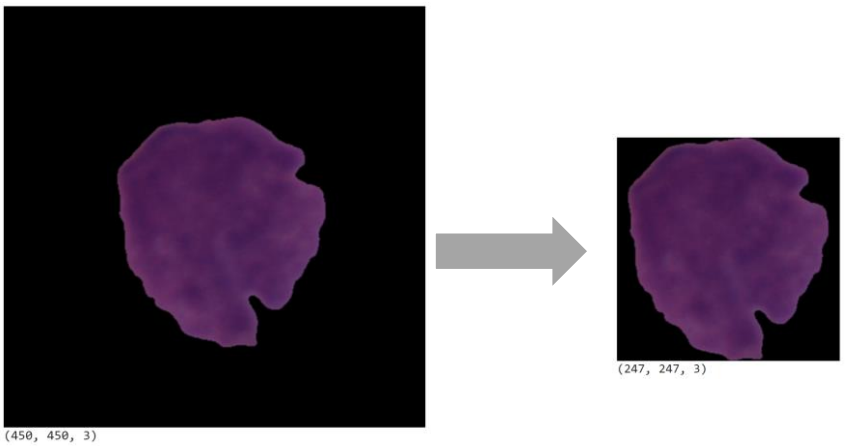


Figure 2. Center cropping

Another thing that may be noticed when analyzing the dataset is that a large part of each image is consists of background of black color. Therefore, the size of the images can be greatly reduced without loss of information by performing center cropping. In order to do this, first the image is converted from RGB to grayscale and then a thresholding method is applied to locate the area of the image that is completely black. Finally, after removing as much of background as possible, black padding is added to make the image square again, as can be seen in Figure 2. In this way, any resizing will not affect the aspect ratio of the cell. As a result of this process, more than half of the images have been reduced to a size smaller than 224×224 and the vast majority are at most 300×300 in size.

Additionally, since it had benefited network performance in [8], we tried applying Contrast-Limited Adaptive Histogram Equalization (CLAHE) to the images. The images are projected into the YUV color space, and CLAHE is performed only on the Y channel. This results in improved contrast and finer detail, as can be seen in Figure 3. However, when testing this technique on our best networks, no significant improvements in performance were noticed.

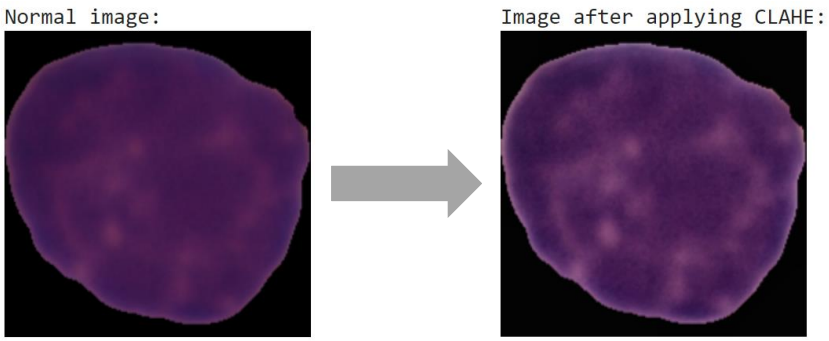


Figure 3. Contrast Limited Adaptive Histogram Equalization applied to an image

3.2. CNN from Scratch

First, we tried to define some architectures from scratch, to propose our own custom architecture for solving the problem. As we know, there are endless possibilities to design a neural network, having many hyperparameters to decide.

The *trial-and-error* approach was applied to find the best architecture, on which we then tried to perform further experiments to improve its performance. We started with a small model that did not fit well and then gradually increase its size until it started to generalize. We will report only the best base architecture, considering some of the best experiments that have been performed on this base.

Regarding the size of the input images, we decided to use 300×300 because of 6396 images in the training set, 6190 are at least 300×300, while the others are smaller. We chose to use 32 as batch size, which is the number of samples considered in each iteration. After the input layer, we performed a normalization with a Rescaling Layer, so that each pixel value was in the range [0,1]. Then we decided to use *4 convolutional layers*, using the *zero-padding* technique, in order to give the same importance to each pixel of the image; in fact, even the borders are relevant having cropped the images as much as possible. For the stride we decided to use the default value of 1, to avoid penalties on the accuracy. As activation functions we used ReLU: $f(x) = \max(0, x)$. As the size of the local receptive fields, we decided to use the default value 3×3. In the first convolutional layer 32 filters are used, and *for each convolutional layer the number of filters doubles*. Max-pooling is applied after each convolutional level, so that small regions are summarized with a single value. Our architecture uses *increasing size for max pooling*, with the size of the pooling increasing in the final levels

(2×2 in the first two layers, 3×3 in the third layer and 5×5 in the last one). This allowed us to decrease the number of network parameters while minimizing accuracy loss. Indeed, we could also increase the stride or avoid zero-padding, but this solution is the one that achieves better performance.

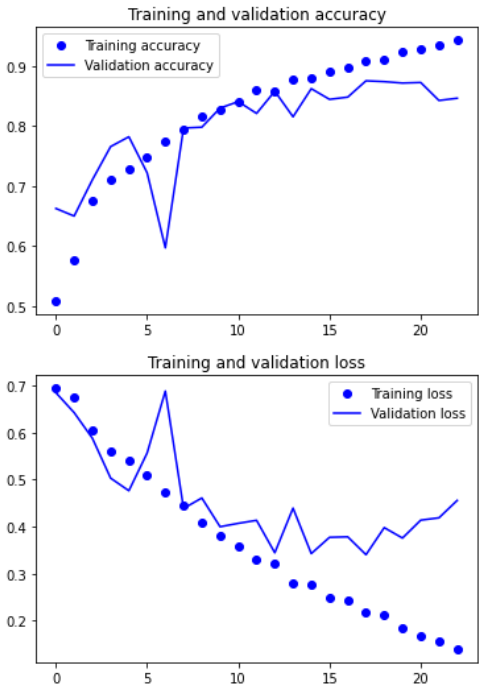
The final levels change in the various experiments made. The only thing that does not change is the output layer, for which we used a dense layer with a single neuron, exploiting a sigmoid function to perform the binary classification.

It is important to point out that as a regularization technique we used the dropout, because in the various experiments it was the one that behaved best with our problem.

3.2.1. One dense layer with 256 neurons and dropout

In this experiment we used a dense layer of 256 neurons with dropout before the output layer. The dropout is used to fight the overfitting, that initially is very high.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.2192	0.9088	0.3403	0.8757




```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 300, 300, 3)]     0

rescaling_1 (Rescaling)      (None, 300, 300, 3)       0

conv2d_4 (Conv2D)            (None, 300, 300, 32)      896

max_pooling2d_4 (MaxPooling  (None, 150, 150, 32)      0
2D)

conv2d_5 (Conv2D)            (None, 150, 150, 64)      18496

max_pooling2d_5 (MaxPooling  (None, 75, 75, 64)        0
2D)

conv2d_6 (Conv2D)            (None, 75, 75, 128)       73856

max_pooling2d_6 (MaxPooling  (None, 25, 25, 128)       0
2D)

conv2d_7 (Conv2D)            (None, 25, 25, 256)       295168

max_pooling2d_7 (MaxPooling  (None, 5, 5, 256)         0
2D)

flatten_1 (Flatten)          (None, 6400)              0

dense_2 (Dense)              (None, 256)               1638656

dropout_1 (Dropout)          (None, 256)               0

dense_3 (Dense)              (None, 1)                 257

=================================================================
Total params: 2,027,329
Trainable params: 2,027,329
Non-trainable params: 0
_________________________________________________________________
```

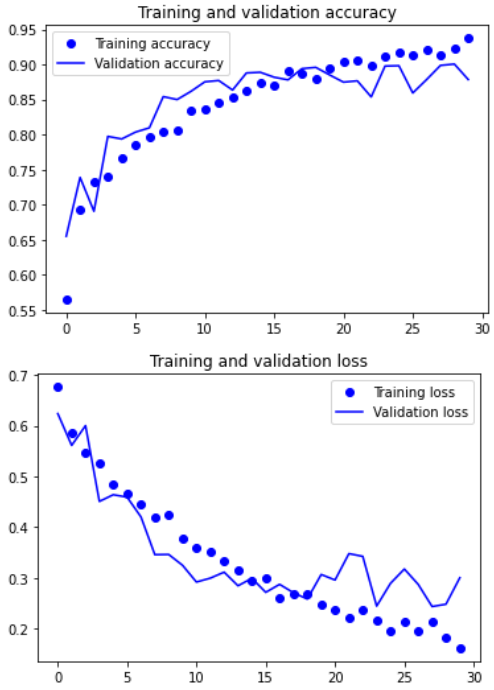
Figure 4. Learning curves and summary of the network

From the learning curves we can see that the model still suffers from overfitting, not being able to generalize well on the validation set. The validation loss is 0.34, that is high.

3.2.2. One dense layer with 256 neurons and two dropout

In this experiment we tried to insert another dropout, just before the dense layer with 256 neurons. So now we have two dropouts.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.2136	0.9140	0.2436	0.8987




```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 300, 300, 3)]     0

rescaling (Rescaling)        (None, 300, 300, 3)       0

conv2d (Conv2D)              (None, 300, 300, 32)      896

max_pooling2d (MaxPooling2D  (None, 150, 150, 32)      0
)

conv2d_1 (Conv2D)            (None, 150, 150, 64)      18496

max_pooling2d_1 (MaxPooling  (None, 75, 75, 64)        0
2D)

conv2d_2 (Conv2D)            (None, 75, 75, 128)       73856

max_pooling2d_2 (MaxPooling  (None, 25, 25, 128)       0
2D)

conv2d_3 (Conv2D)            (None, 25, 25, 256)       295168

max_pooling2d_3 (MaxPooling  (None, 5, 5, 256)         0
2D)

flatten (Flatten)            (None, 6400)              0

dropout (Dropout)            (None, 6400)              0

dense (Dense)                (None, 256)               1638656

dropout_1 (Dropout)          (None, 256)               0

dense_1 (Dense)              (None, 1)                 257

=================================================================
Total params: 2,027,329
Trainable params: 2,027,329
Non-trainable params: 0
_________________________________________________________________
```

Figure 5. Learning curves and summary of the network

We can see that now the network reaches a better validation loss, more precisely 0.2436. This is an excellent result; the network can generalize the problem well. Now we will analyze the performance on the test set of this network.

Precision		Recall		Weighted F1-Score
Normal	*Cancer*	*Normal*	*Cancer*	
0.8754	0.9230	0.8376	0.9422	0.9076

The neural network has a high precision and recall for the positive class, which is the most important factor. Indeed, a false negative is much more dangerous than a false positive. And still the performance on the negative class is satisfactory, in line with *state-of-the-art*. The AUC is 0.95, the test is highly accurate.

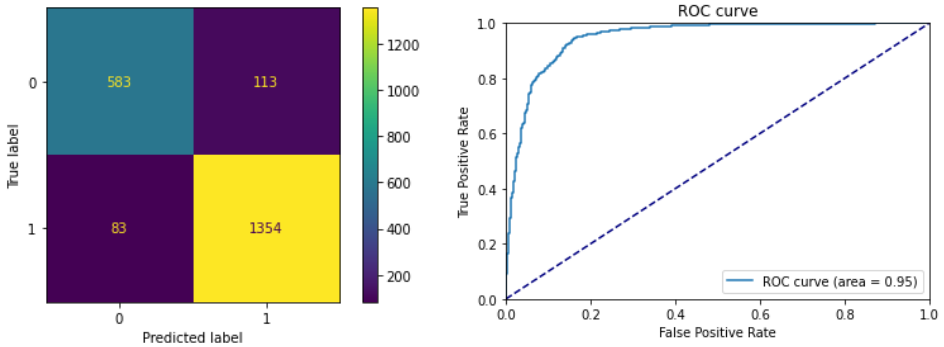
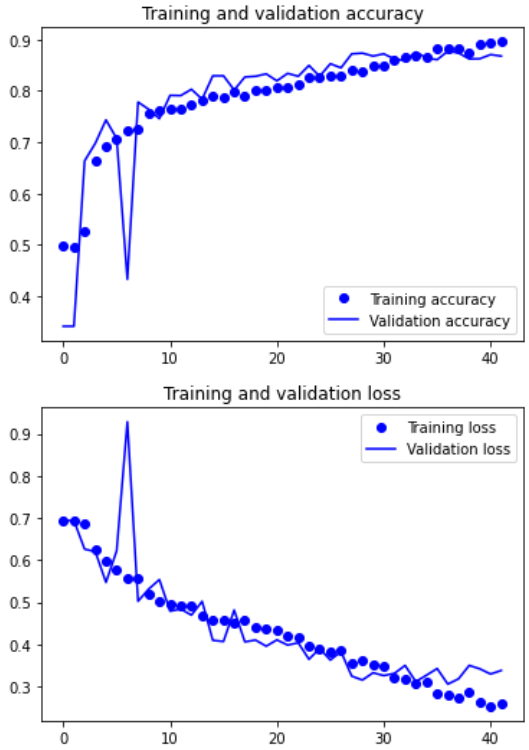


Figure 6. Confusion matrix and ROC curve

3.2.3. One dense layer with 128 neurons and dropout

In this experiment we tried to decrease the number of neurons in the dense layer to fight the slight overfitting. The network is the same as above, but now the dense layer has 128 neurons.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.2814	0.8818	0.3056	0.8771




```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 300, 300, 3)]     0
rescaling (Rescaling)        (None, 300, 300, 3)       0
conv2d (Conv2D)              (None, 300, 300, 32)      896
max_pooling2d (MaxPooling2D  (None, 150, 150, 32)      0
)
conv2d_1 (Conv2D)            (None, 150, 150, 64)      18496
max_pooling2d_1 (MaxPooling  (None, 75, 75, 64)        0
2D)
conv2d_2 (Conv2D)            (None, 75, 75, 128)       73856
max_pooling2d_2 (MaxPooling  (None, 25, 25, 128)       0
2D)
conv2d_3 (Conv2D)            (None, 25, 25, 256)       295168
max_pooling2d_3 (MaxPooling  (None, 5, 5, 256)         0
2D)
flatten (Flatten)            (None, 6400)              0
dropout (Dropout)            (None, 6400)              0
dense (Dense)                (None, 128)               819328
dropout_1 (Dropout)          (None, 128)               0
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,207,873
Trainable params: 1,207,873
Non-trainable params: 0
_________________________________________________________________
```

Figure 7. Learning curves and summary of the network

We can see that the generalization capability is good also in this experiment, but the network reaches a slightly worse result and learning is much slower. To note even better the differences with the previous network, we show the results on the test set.

Precision		Recall		Weighted F1-Score
Normal	*Cancer*	*Normal*	*Cancer*	
0.8682	0.8638	0.6911	0.9492	0.8605

The network has still a high recall on the positive class, but a lower precision. Also, the results on the negative class are not satisfactory. The AUC is 0.92, so the test is highly accurate.

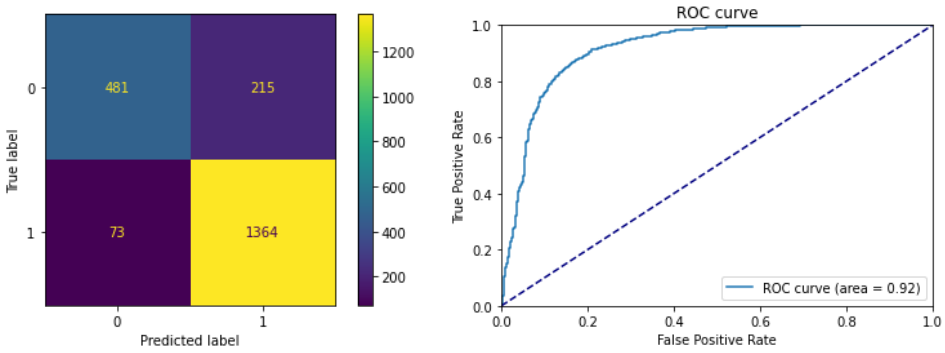


Figure 8. Confusion matrix and ROC curve

3.2.4. Two dense layers

In this experiment we tried to increase the number of dense layers, increasing the capacity of the network. We have now two dense layers, with 256 neurons for each of them.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.3492	0.8490	0.3744	0.8537

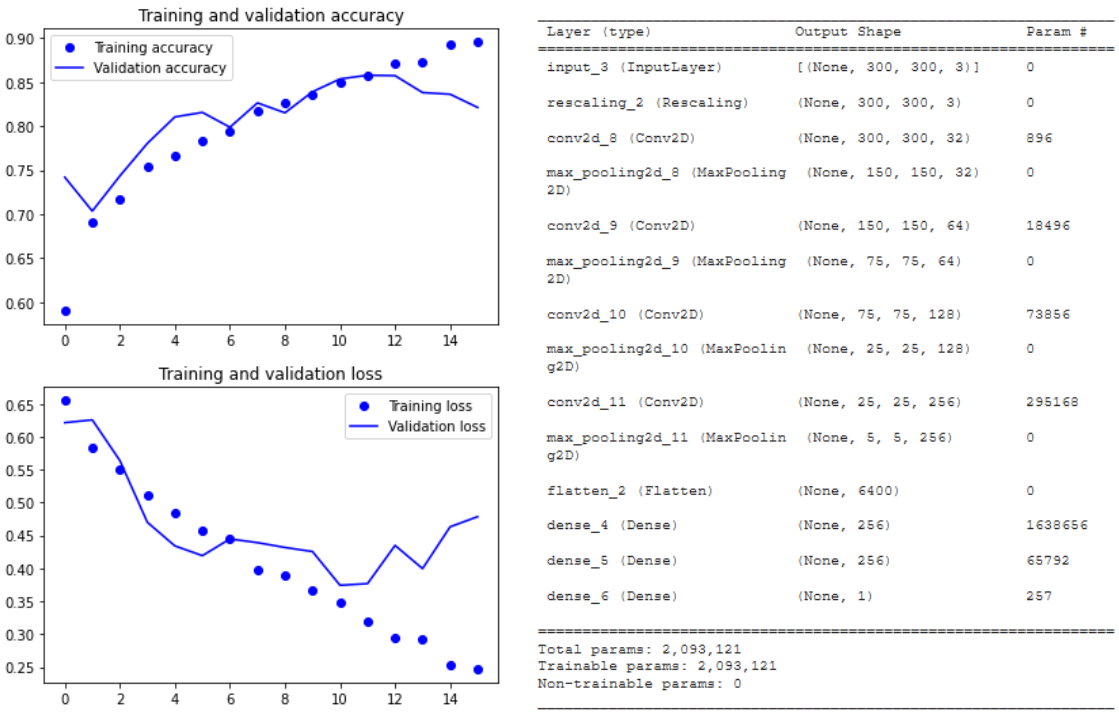


Figure 9. Learning curves and summary of the network

This approach has not led to improvements. The network suffers from even greater overfitting. We also tried other combinations for the number of neurons in the two dense layers, such as 128 neurons in the first layer and 256 in the second one, but the results did not improve.

3.3. Pre-trained Models

The results obtained with CNN from scratch are already quite satisfactory, especially when compared to the state of the art. However, we think we can achieve even better performance by exploiting transfer learning.

In fact, having a relatively small training set available, it is difficult for us to develop powerful networks that can generalize effectively. This task can be simplified by using pre-trained networks. In particular, for each of the networks mentioned below, we tried fitting the model to our problem with both feature extraction and fine-tuning, using a *trial-and-error* approach.

VGG16 is a convolutional neural network model proposed in 2014 by K. Simonyan and A. Zisserman from the University of Oxford in the paper "*Very Deep Convolutional Networks for Large-Scale Image Recognition*" [11].

ResNet50, short for Residual Network with 50 layers, is a specific type of neural network that was introduced in 2015 by K. He, X. Zhang, S. Ren and J. Sun in their paper "*Deep Residual Learning for Image Recognition"* [12].

MobileNetv2 is a simple but efficient and not very computationally intensive convolutional neural network that was proposed in 2019 by M. Sandler, A. G. Howard, M. Zhu & al. in the paper "*MobileNetV2: Inverted Residuals and Linear Bottlenecks"* [13].

3.3.1. VGG16

The original architecture of VGG16 is shown in Figure 10. The convolutional base consists of five blocks with 2 or 3 convolutional layers whose filters have a size of 3×3. The stack of convolutional blocks is followed by a densely connected classifier consisting of three layers. The final layer is a 1000-node softmax layer. All hidden layers are equipped with the rectification (ReLU). Furthermore, between blocks max-pooling is performed on a 2×2 window, with stride 2.

The network expects an input of fixed dimension of 224×224×3 and generates as output a vector of 1000 probabilities corresponding to the 1000 classes in the ImageNet dataset. The input image is assigned to the class with the maximum probability.

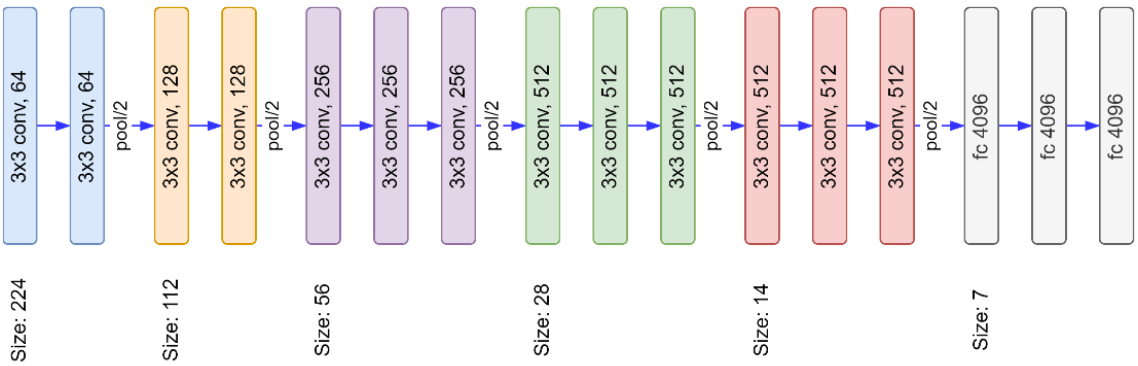


Figure 10. VGG16 architecture

Our experiments on the VGG16 architecture were performed in three stages. First of all, we performed a simple feature extraction by taking the convolutional base of the network as it is and training a new classifier on top of its output. Next,

taking into consideration the combination we thought was the best for the classifier, we fine-tuned the convolutional base, gradually unfreezing the various layers, starting with those closest to the output.

With neither of these techniques we were able to achieve great results, so we proceeded with the third stage. Since our dataset is small and very different from the original dataset, we though to remove the last convolutional block and to perform feature extraction and fine-tuning with only the underlying architecture. The idea is that, in this way, we can leverage blocks capable of detecting features that are more generic and less related to the original training dataset.

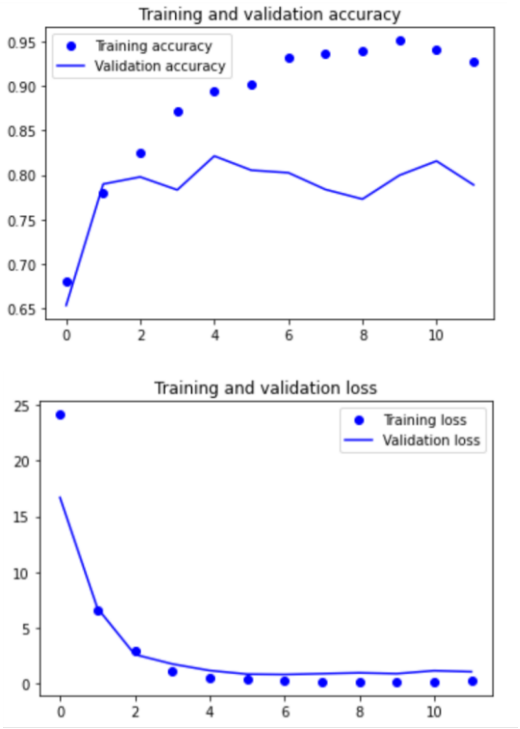
3.3.1.1 Feature Extraction

The very first thing we tried was to use the convolutional base of the architecture solely as a feature extractor. Since we are dealing with medical images, which are very different from the images in ImageNet, we did not expect to achieve good results, and so it was. In particular, despite trying different combinations for the classifier and different regularization techniques, the model was only able to perform well on the training set, showing an inability to generalize.

3.3.1.1.1 One fully connected layer with dropout

In this experiment we used a dense layer of 256 neurons with dropout before the output layer. This seems to be the best configuration but, as anticipated and as can be seen from the results and the learning curves, even if the model succeeds to learn from the training set, it does not perform well on that of validation. In fact, while the training loss is 0.19, the validation loss is 0.7938, which is very high.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.1900	0.9314	0.7938	0.8025




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_1   (None, 224, 224, 3)      0
(SlicingOpLambda)

tf.nn.bias_add_1 (TFOpLambd  (None, 224, 224, 3)      0
a)

vgg16 (Functional)           (None, 7, 7, 512)         14714688

flatten_1 (Flatten)          (None, 25088)             0

dense_2 (Dense)              (None, 256)               6422784

dropout (Dropout)            (None, 256)               0

dense_3 (Dense)              (None, 1)                 257

=================================================================
Total params: 21,137,729
Trainable params: 6,423,041
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 11. Learning curves and summary of the network

3.3.1.2 Fine Tuning

Since the highest layers of the network have most likely learned to recognize features that are more specific to the original dataset, we though that performing fine-tuning might be beneficial to the performance of the model. However, unfortunately, we were unable to achieve satisfactory results.

In fact, even unfreezing few layers out of concern of increasing too much the number of layers and consequently the risk of overfitting, all the models obtained were unable to generalize and achieved very poor performances on the validation set. As an example, we show below one of the experiments performed.

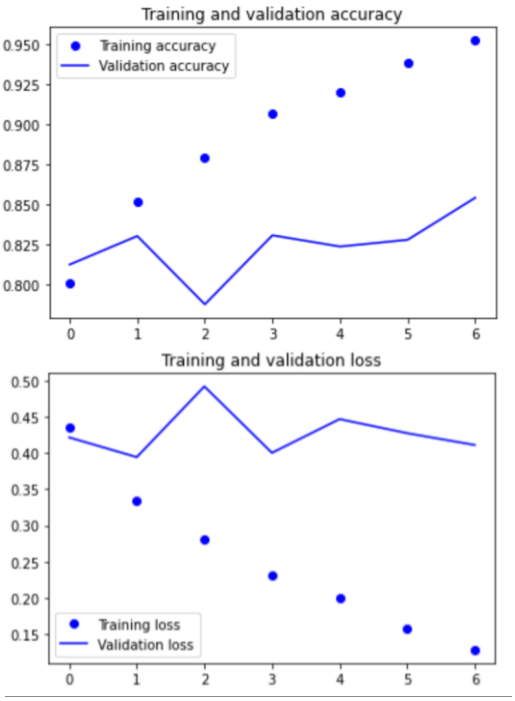
3.3.1.2.1 Last three layers

In this experiment we used the classifier from the previous experiment, we unfrozen the last three convolutional layers of the convolutional base and we trained the resulting model with our training set with a very low learning rate.

Training loss	**Training accuracy**	**Validation loss**	**Validation accuracy**
0.347	0.8518	0.3941	0.8302

Although at first the learning curve on the validation set seems to almost follow the curve on the training set, the peak is reached immediately at the second epoch, and the curve does not rise again. Thus, the model seems to suffer from overfitting, as performance on the training set improves with each epoch, while performance on the validation set remains around the same values.

The fact that the network was not able to generalize in any of the experiments we performed, despite adding regularization techniques and unfreezing a few layers at a time, prompted us to move forward with the next stage: removing the last convolutional block.




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        [(None, 224, 224, 3)]     0

tf.__operators__.getitem_11  (None, 224, 224, 3)      0
 (SlicingOpLambda)

tf.nn.bias_add_11 (TFOpLamb  (None, 224, 224, 3)      0
da)

vgg16 (Functional)           (None, 7, 7, 512)         14714688

dropout (Dropout)            (None, 7, 7, 512)         0

flatten_9 (Flatten)          (None, 25088)             0

dense_20 (Dense)             (None, 256)               6422784

dropout_1 (Dropout)          (None, 256)               0

dense_21 (Dense)             (None, 1)                 257

=================================================================
Total params: 21,137,729
Trainable params: 13,502,465
Non-trainable params: 7,635,264
_________________________________________________________________
```

Figure 12. Learning curves and summary of the network

3.3.1.3 Removing Last Block

As already anticipated, after running several experiments with the whole convolutional base, we decided to remove block number 5. Again, we performed several experiments in both feature extraction and fine-tuning. Below, we report the ones that gave us the best results.

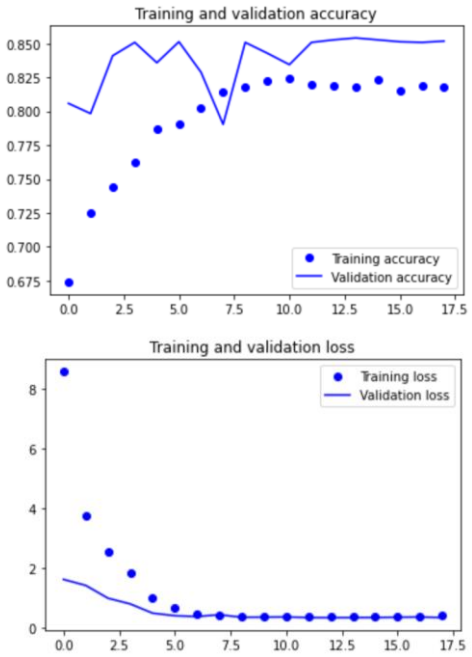
3.3.1.3.1 Feature Extraction: Global Average Pooling + one fully connected layer with dropout

In this experiment we used a dense layer of 256 neurons with dropout before the output layer. The dropout, as usual, is used to fight overfitting. Since this time, we are putting the classifier on top of an intermediate layer of the original architecture, the number of parameters is very large. In order to avoid an "explosion" in the number of parameters to be computed, we applied global average pooling before the dense layer.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.4023	0.8185	0.3484	0.8527

As can be seen from the results and learning curves, performance is slightly better on the validation set than on the training set. This could be due to the presence of dropout, which slows down learning but does not affect performance on the validation set.

In addition, this could also mean that we are somehow underfitting, given the small number of parameters obtained as a result of the pooling layer. Since we have fine-tuning in mind, this problem should be solved by unfreezing the underlying layers and thus increasing the number of parameters.



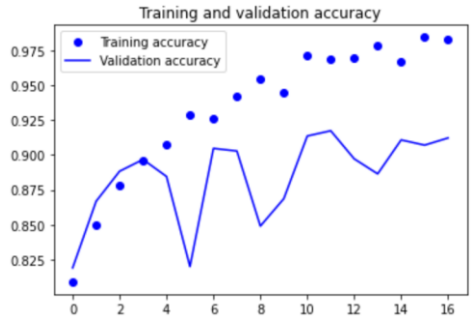

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_4   (None, 224, 224, 3)      0
(SlicingOpLambda)

tf.nn.bias_add_4 (TFOpLambd  (None, 224, 224, 3)      0
a)

sequential_2 (Sequential)    (None, 14, 14, 512)       7635264

global_average_pooling2d_2   (None, 512)               0
(GlobalAveragePooling2D)

dense_4 (Dense)              (None, 256)               131328

dropout_2 (Dropout)          (None, 256)               0

dense_5 (Dense)              (None, 1)                 257

=================================================================
Total params: 7,766,849
Trainable params: 131,585
Non-trainable params: 7,635,264
_________________________________________________________________
```

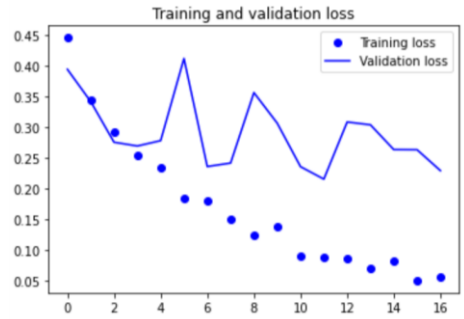
Figure 13. Learning curves and summary of the network

3.3.1.3.2 Fine-tuning: last three blocks

Using the classifier from the previous experiment, we gradually unfrozen the convolutional layers and performed fine-tuning, down to three blocks (from block 2 to block 4). For fine-tuning, we decided to use a very low learning rate in order to limit the magnitude on changes made to the original weights.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.0879	0.9689	0.2152	0.9174






```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_4   (None, 224, 224, 3)      0
(SlicingOpLambda)

tf.nn.bias_add_4 (TFOpLambd  (None, 224, 224, 3)      0
a)

sequential_2 (Sequential)    (None, 14, 14, 512)       7635264

global_average_pooling2d_2   (None, 512)               0
(GlobalAveragePooling2D)

dense_4 (Dense)              (None, 256)               131328

dropout_2 (Dropout)          (None, 256)               0

dense_5 (Dense)              (None, 1)                 257

=================================================================
Total params: 7,766,849
Trainable params: 7,728,129
Non-trainable params: 38,720
_________________________________________________________________
```

Figure 14. Learning curves and summary of the network

As can be seen from the learning curves, this model achieves very good results on the training set, but this trend is followed only up to a certain point for the validation set. This could be symptomatic of overfitting and could leave room for further improvement. Unfortunately, with our experiments, despite trying different combinations of hyperparameters and different regularization techniques, we were unable to obtain better results.

In any case, analyzing the results of the model on the test set, it is possible to see that these are satisfactory. As a matter of fact, the model is able to achieve a weighted F1-score of 0.9183 and has very high precision and recall for the positive class (which we are interested in, because of the cost of a false negative). Furthermore, the AUC is 0.96, which is close to 1 and indicates that the test is highly accurate.

Precision		Recall		Weighted F1-Score
Normal	*Cancer*	*Normal*	*Cancer*	
0.8980	0.9282	0.8477	0.9534	0.9183

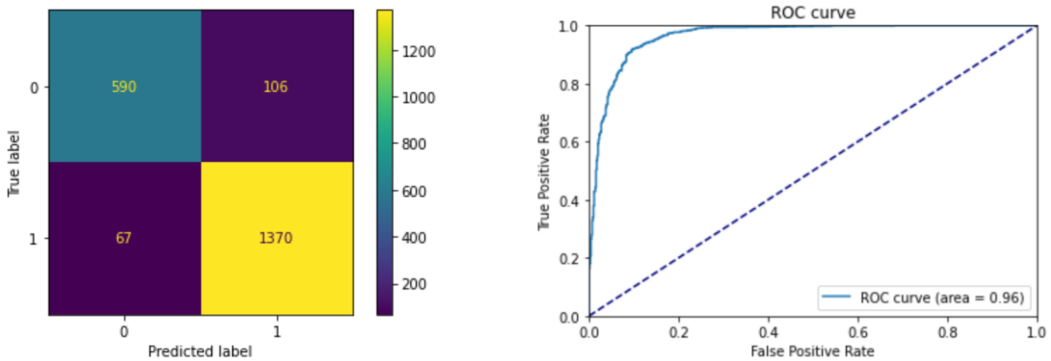


Figure 15. Confusion matrix and ROC curve

3.3.2. ResNet-50

ResNet-50 use a convolutional neural network (CNN) that is 50 layers deep. ResNet has many variants that run on the same concept but have different numbers of layers. It was created with the aim of tackling the problem of degradation in deep neural networks, with the use of residual blocks. In fact, while the number of stacked layers can enrich the features of the model, when the layers of a neural network increase, the accuracy level may get saturated and slowly degrade after a point. With residual blocks the ResNet can improve the accuracy, with the concept of "skip connections", that permit alternate shortcut for the gradient to pass through. The original architecture of ResNet-50 is shown below. The convolutional base consists of five blocks with convolutional layers whose filters have non decreasing sizes. The stack of convolutional blocks is followed by a densely connected classifier. The final layer is a 1000-node softmax layer.

The network expects an input of fixed dimension of 224×224×3 and generates as output a vector of 1000 probabilities corresponding to the 1000 classes in the ImageNet dataset. The input image is assigned to the class with the maximum probability.

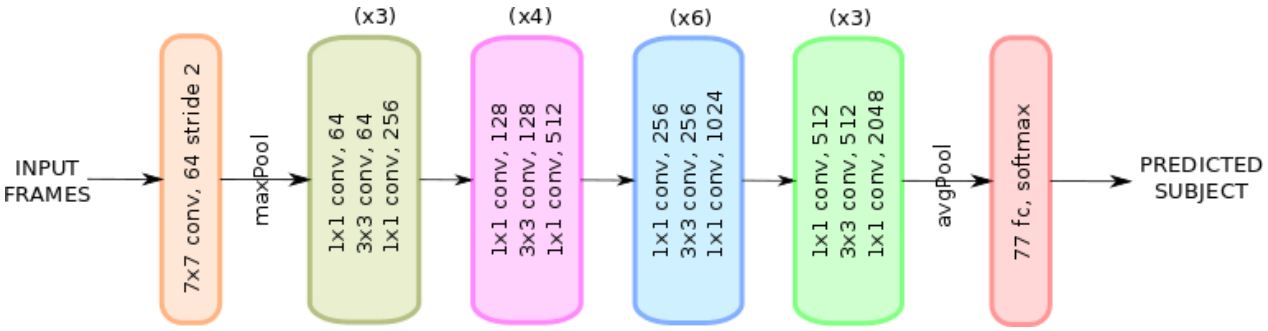


Figure 16. ResNet-50 architecture

Again, we performed our experiments in three stages, starting from a simple feature extraction taing the convolutional base of the network with new classifiers on top of its output. After, we took in consideration the best models and we tried to fine-tune the convolutional base, gradually unfreezing the layers. The results obtained were not satisfactory, and this was mainly due to the fact that our dataset is small and very different from ImageNet. Also in this case, we removed the last convolutional block and performed again feature extraction and fine-tuning. Thanks to this trick, we were able to get some great results, which even exceed the *state-of-the-art*.

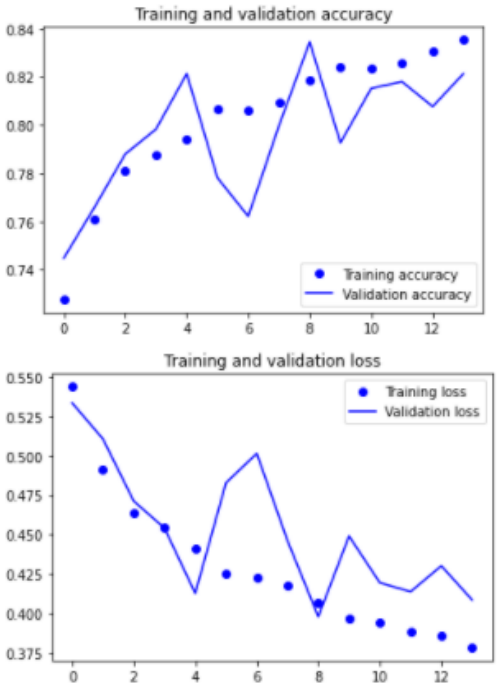
3.1.1.1. Feature Extraction

The very first thing we tried was to use the convolutional base of the architecture as a feature extractor. Since we are dealing with medical images, which are very different from the images in ImageNet, we did not expect to achieve good results, and so it was. In particular, despite trying different experiments, the models show inability to generalize.

3.1.1.1.1 Global average pooling

In some of the most promising experiments, we used global average pooling instead of Flatten, a more modern approach. We will report just the model that performed best after fine tuning.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.409	0.8185	0.3979	0.8344




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_1   (None, 224, 224, 3)       0
(SlicingOpLambda)

tf.nn.bias_add_1 (TFOpLambd  (None, 224, 224, 3)       0
a)

resnet50 (Functional)        (None, 7, 7, 2048)        23587712

global_average_pooling2d_1   (None, 2048)              0
(GlobalAveragePooling2D)

dense_3 (Dense)              (None, 1)                 2049

=================================================================
Total params: 23,589,761
Trainable params: 2,049
Non-trainable params: 23,587,712
_________________________________________________________________
```

Figure 17. Learning curves and summary of the network

Up to a certain point the loss curve on validation correctly follows the training curve, then there starts to be overfitting. Several experiments have been performed to fight this behavior, but then we decided to switch to fine-tuning and see what improvements it brings.

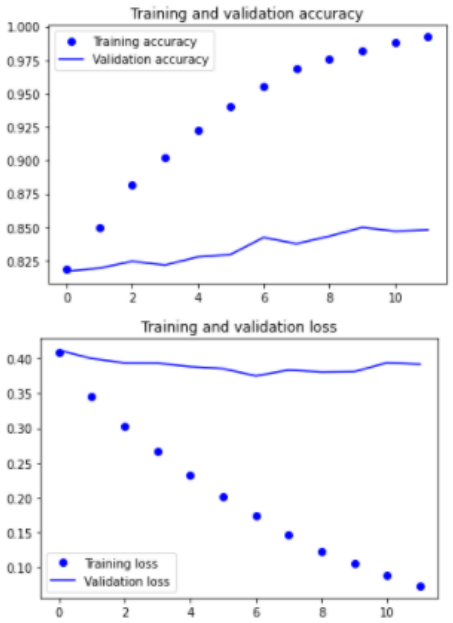
3.1.1.2. Fine Tuning

We thought that performing fine-tuning might be beneficial to the performance of the model. However, we were unable to achieve satisfactory results. We could not unfreeze too many levels, otherwise the number of parameters would explode, with risk of overfitting.

3.1.1.2.1 Unfreeze last three layers

We tried to unfreeze the last three convolutional layers. There have been improvements, but not enough.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.1735	0.9553	0.3750	0.8424




```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem_1   (None, 224, 224, 3)      0
 (SlicingOpLambda)

 tf.nn.bias_add_1 (TFOpLambd  (None, 224, 224, 3)      0
 a)

 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d_1   (None, 2048)             0
 (GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 1)                 2049

=================================================================
Total params: 23,589,761
Trainable params: 4,467,713
Non-trainable params: 19,122,048
_________________________________________________________________
```

Figure 18. Learning curves and summary of the network

As already announced the results are not satisfactory, this is mainly noticeable from the performance on the test set. In general, the validation loss is still high after fine tuning and the results on the negative class are not satisfactory, while the results on the positive class are decent. The AUC is 0.91, so the test is highly accurate.

Precision		**Recall**		**Weighted F1-Score**
Normal	*Cancer*	*Normal*	*Cancer*	
0.7776	0.9056	0.8089	0.8880	0.8628

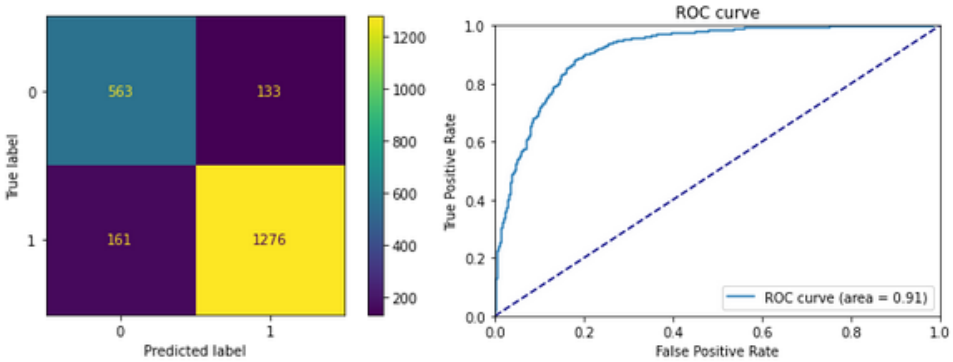


Figure 19. Confusion matrix and ROC curve

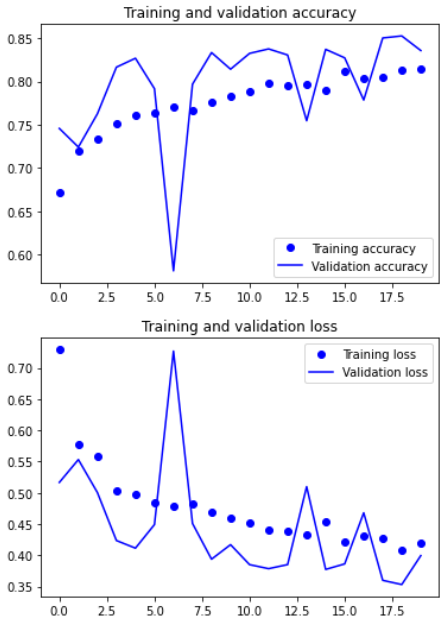
3.3.2.1 Removing Last Block

Now we will consider some of the results obtained removing the 5-th block. With this modification the output of our base network is 14×14×1024, too much to use Flatten; so, we decided to use a more modern approach, based on *Global Average Pooling*. As always, we report only the most significant experiments.

3.3.2.1.1 Feature Extraction: one dense layer with 256 neurons and dropout

In this experiment we tried a dense layer with 256 neurons after the global average pooling layer, with a dropout before the output layer.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.4074	0.8127	0.3532	0.8527




```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_1   (None, 224, 224, 3)       0
(SlicingOpLambda)

tf.nn.bias_add_1 (TFOpLambd  (None, 224, 224, 3)       0
a)

base (Functional)            (None, 14, 14, 1024)      8589184

global_average_pooling2d_1   (None, 1024)              0
(GlobalAveragePooling2D)

dense_1 (Dense)              (None, 256)               262400

dropout (Dropout)            (None, 256)               0

dense_2 (Dense)              (None, 1)                 257

=================================================================
Total params: 8,851,841
Trainable params: 262,657
Non-trainable params: 8,589,184
_________________________________________________________________
```

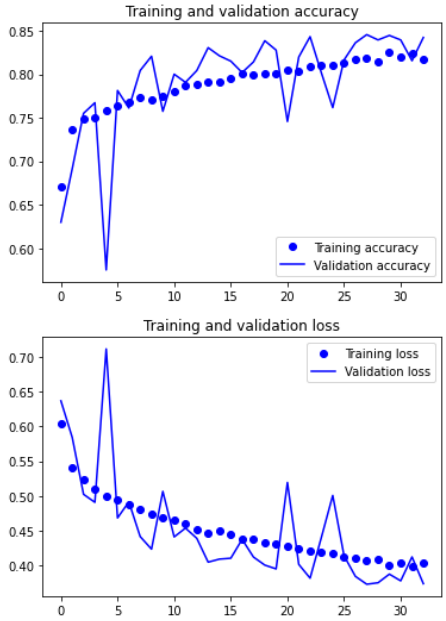
Figure 20. Learning curves and summary of the network

We can see that there is underfitting, but we still need to fine-tune so the problem will lower. The presence of dropout helps in fighting overfitting, but results in a slowing of the training curve. In general, the loss is still high and the accuracy not very satisfactory, but this model will then be considered with fine-tuning to see what improvements it brings.

3.3.2.1.2 Feature Extraction: one dense layer with 512 neurons

In this experiment we tried only a dense layer with 512 neurons after the global average pooling layer. The idea is to fight underfitting increasing the number of neurons and removing the dropout. We tried also to change the optimizer, to fight curve instability. We got more satisfactory results with RMSProp, with 0.00005 as learning rate.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.4075	0.8186	0.3726	0.8457




```
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_4   (None, 224, 224, 3)       0
(SlicingOpLambda)

tf.nn.bias_add_4 (TFOpLambd  (None, 224, 224, 3)       0
a)

base (Functional)            (None, 14, 14, 1024)      8589184

global_average_pooling2d_3   (None, 1024)              0
(GlobalAveragePooling2D)

dense_9 (Dense)              (None, 512)               524800

dense_10 (Dense)             (None, 1)                 513

=================================================================
Total params: 9,114,497
Trainable params: 525,313
Non-trainable params: 8,589,184
_________________________________________________________________
```

Figure 21. Learning curves and summary of the network

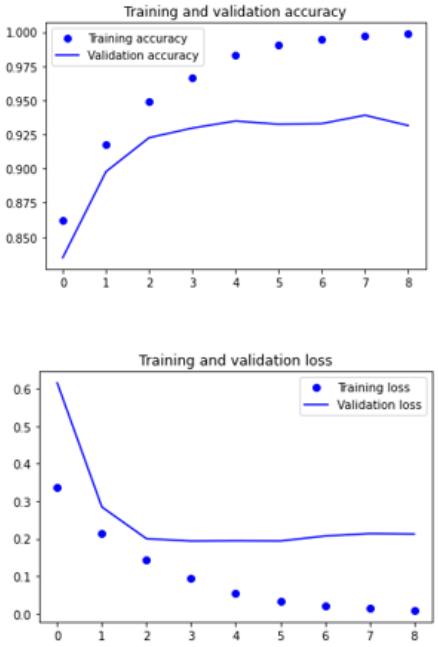
We can still see a general underfitting, that could not be solved even by increasing network capacity during subsequent experiments. The learning curves are more stable, but the validation loss has not improved. To fight the remaining underfitting, we must move to the fine-tuning phase, trying to adapt the base network to our application context.

3.3.2.1.3 Fine tuning: one dense layer with 256 neurons and dropout

We tried a lot of experiments to understand how much layers to unfreeze. We started with just some layers of the final block, and after we tried to unfreeze entire blocks. At each step we noticed improvement, so we continued to unfreeze layers. Finally, we noticed an improvement in unfreezing all the layers of the base network. For fine-tuning we used a smaller learning rate, more precisely 0.00001.

Training loss	**Training accuracy**	**Validation loss**	**Validation accuracy**
0.0953	0.9662	0.1937	0.9296

We can see that the network has a good generalization capability, with a validation loss of 0.19. This is an astonishing result, that leads to really good results also in the test set.




```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_1   (None, 224, 224, 3)       0
(SlicingOpLambda)

tf.nn.bias_add_1 (TFOpLambd  (None, 224, 224, 3)       0
a)

base (Functional)            (None, 14, 14, 1024)      8589184

global_average_pooling2d_1   (None, 1024)              0
(GlobalAveragePooling2D)

dense_1 (Dense)              (None, 256)               262400

dropout (Dropout)            (None, 256)               0

dense_2 (Dense)              (None, 1)                 257

=================================================================
Total params: 8,851,841
Trainable params: 8,821,249
Non-trainable params: 30,592
_________________________________________________________________
```

Figure 22. Learning curves and summary of the network

Precision		**Recall**		**Weighted F1-Score**
Normal	*Cancer*	*Normal*	*Cancer*	
0.9512	0.9386	0.8678	0.9784	0.9416

The positive class is classified very well, and we have very good results for the negative class as well. The AUC is 0.98, so the test is highly accurate.

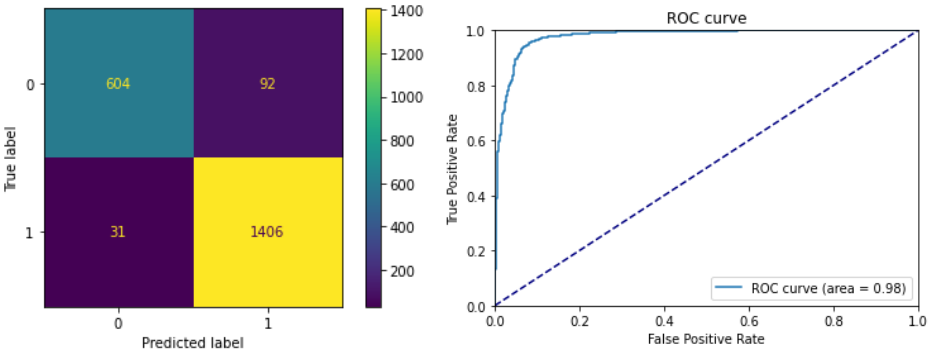
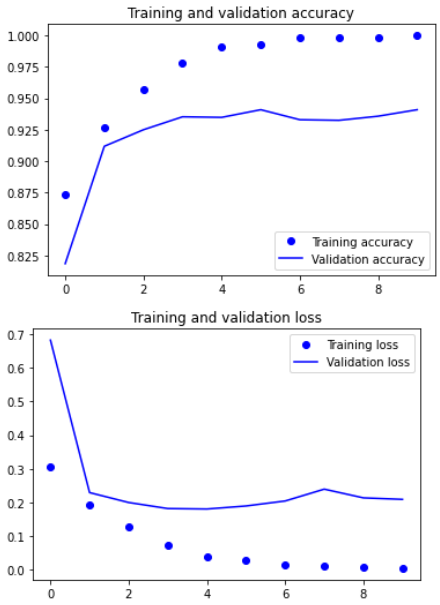


Figure 23. Confusion matrix and ROC curve

3.3.2.1.4 Fine tuning: one dense layer with 512 neurons

In this experiment we tried to fine-tune also the second good model, and also in this case we obtained good results. In this case we used RMSProp as optmizer.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.0390	0.9906	0.1801	0.9348




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_4   (None, 224, 224, 3)       0
(SlicingOpLambda)

tf.nn.bias_add_4 (TFOpLambd  (None, 224, 224, 3)       0
a)

base (Functional)            (None, 14, 14, 1024)      8589184

global_average_pooling2d_3   (None, 1024)              0
(GlobalAveragePooling2D)

dense_9 (Dense)              (None, 512)               524800

dense_10 (Dense)             (None, 1)                 513

=================================================================
Total params: 9,114,497
Trainable params: 9,083,905
Non-trainable params: 30,592
_________________________________________________________________
```

Figure 24. Learning curves and summary of the network

Also, in this experiment the model has a good generalization capability, and we reached a validation loss of 0.18, that is really low. In fact, this model also performs very well on the test set.

Precision		Recall		Weighted F1-Score
Normal	*Cancer*	*Normal*	*Cancer*	
0.9214	0.9564	0.9095	0.9624	0.9451

This model has similar performance compared with the last one, but it is interesting to point out that this model is more accurate on the negative class, while the last one was more accurate on the positive class. In general, we give more importance to the positive class, but inside an ensemble classifier can be very useful also this model, to classify optimally also the negative class.

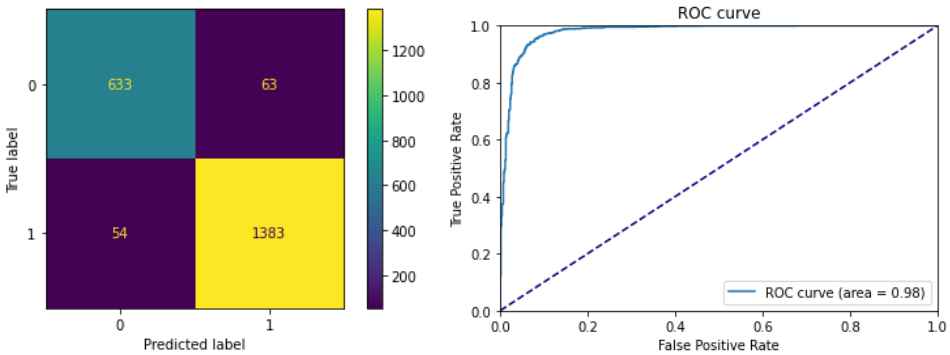


Figure 25. Confusion matrix and ROC curve

3.3.3. MobileNetV2

MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. As a whole, the architecture contains an initial fully convolution layer with 32 filters, followed by original basic blocks named bottleneck. These blocks are then followed by a 1×1 convolution layer with an average pooling layer. The last layer is the classification layer. ReLU is used as non-linearity because of its robustness when used with low-precision computation. Furthermore, kernel size 3×3 is always used as is standard for modern networks, and dropout and batch normalization were utilized during training. The output is a vector of 1000 probabilities corresponding to the 1000 classes in the ImageNet dataset. The input image is assigned to the class with the maximum probability.

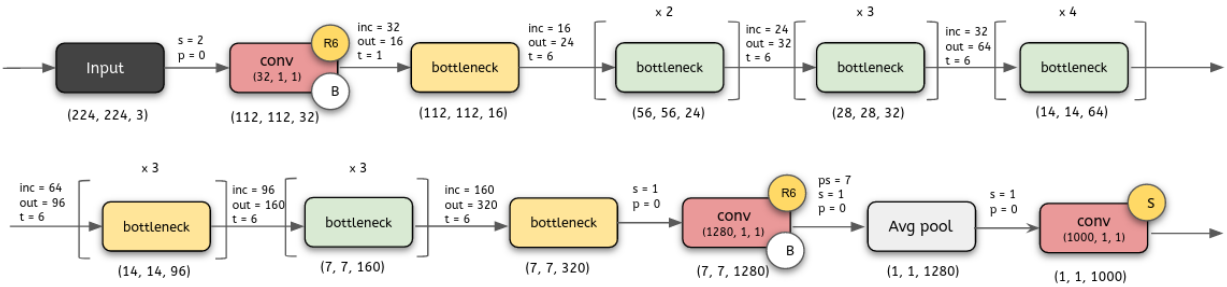


Figure 26. MobileNetV2 architecture

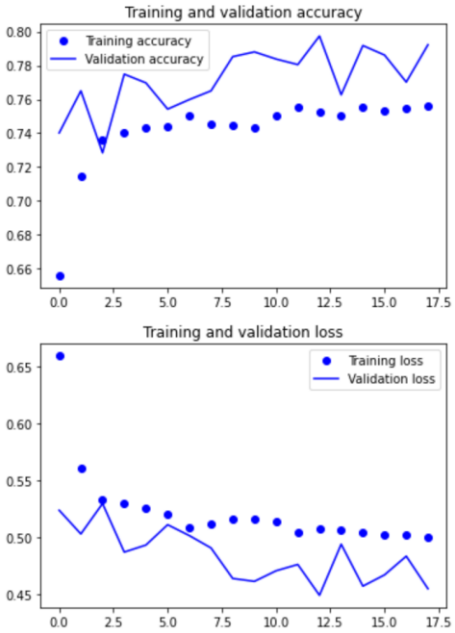
In terms of the experiments performed, also for MobileNetV2 we followed the steps addressed for the others pre-trained networks. We decided to experiment with this architecture both because it produced good results in [7] and because the number of parameters is low, thus it should be suitable for our small dataset.

3.1.1.3. Feature Extraction

Inspired by the setup used in [7], in this experiment the convolutional base is followed by a *Global Average Pooling* layer, dropout and a dense layer with sigmoid activation.

Training loss	**Training accuracy**	**Validation loss**	**Validation accuracy**
0.5069	0.7523	0.4488	0.7974

Among the various configurations that we have experimented, indeed this one seems to be the best but still fails to get good results. In fact, as we can see from the learning curves, the network does not seem to be able to learn from the training set. Nevertheless, we tried to improve the performance of this model with fine tuning.




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_17 (InputLayer)        [(None, 224, 224, 3)]     0

tf.math.truediv_12 (TFOpLam  (None, 224, 224, 3)       0
bda)

tf.math.subtract_12 (TFOpLa  (None, 224, 224, 3)       0
mbda)

mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)        2257984
ional)

global_average_pooling2d_3   (None, 1280)              0
(GlobalAveragePooling2D)

dropout_5 (Dropout)          (None, 1280)              0

dense_16 (Dense)             (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

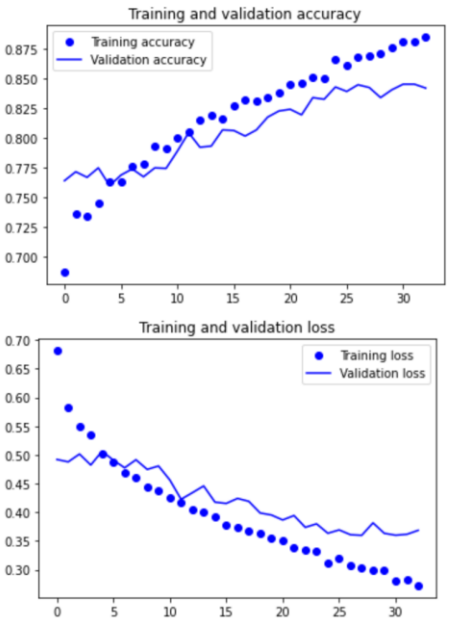
Figure 27. Learning curves and summary of the network

3.3.3.1 Fine Tuning

3.3.3.1.1 Last Bottleneck Block

In this experiment, starting with the setup shown in the previous paragraph, we unfroze the layers down to the last bottleneck block. By looking at the learning curves, we can see that the network seems to learn well from the training set, although very slowly, and the results on the validation set seem to go hand in hand.

Training loss	Training accuracy	Validation loss	Validation accuracy
0.3029	0.8690	0.3598	0.8424




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_17 (InputLayer)        [(None, 224, 224, 3)]     0

tf.math.truediv_12 (TFOpLam  (None, 224, 224, 3)       0
bda)

tf.math.subtract_12 (TFOpLa  (None, 224, 224, 3)       0
mbda)

mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)        2257984
ional)

global_average_pooling2d_3   (None, 1280)              0
(GlobalAveragePooling2D)

dropout_5 (Dropout)          (None, 1280)              0

dense_16 (Dense)             (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 887,361
Non-trainable params: 1,371,904
_________________________________________________________________
```

Figure 28. Learning curve and summary of the network

The problem, however, is that the model fails to achieve excellent performance. In fact, already on the training and validation set it is unable to exceed 87% accuracy and has a rather high loss. When we evaluated the performance of the model on the test set, we could confirm that the model falls short of its predecessors, obtaining a weighted F1-score of only 0.8425 and an AUC of 0.90.

Precision		**Recall**		**Weighted F1-Score**
Normal	*Cancer*	*Normal*	*Cancer*	
0.7579	0.8836	0.7601	0.8824	0.8425

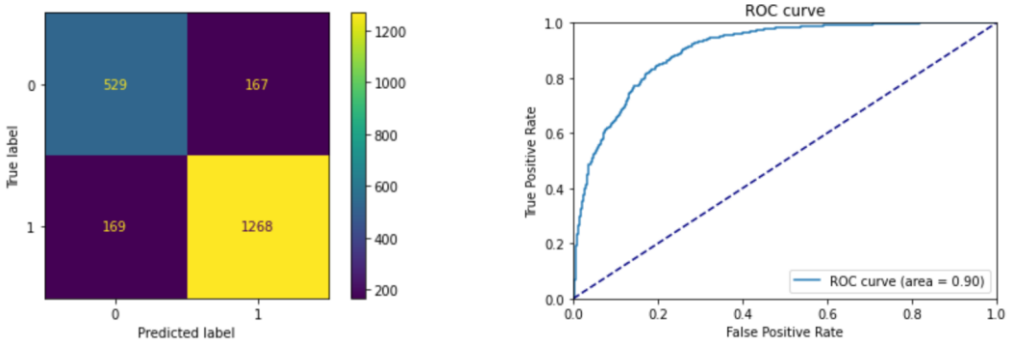
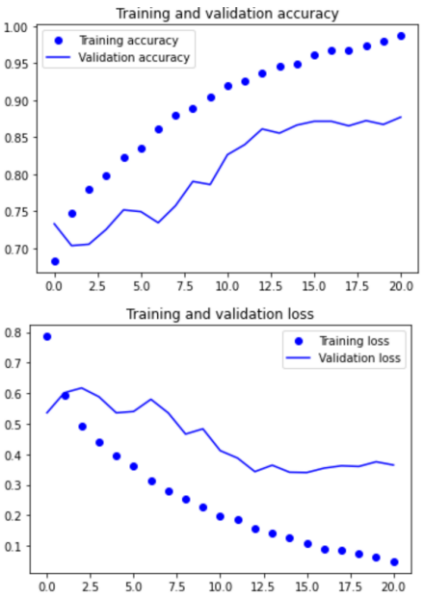


Figure 29. Confusion matrix and ROC curve

3.3.3.1.2 Last 7 Bottleneck Blocks

In an attempt to improve on the results obtained in the previous experiment, and again inspired by the model in [7], we tried in this experiment to unfreeze additional layers, until we reached the tenth bottleneck block. In this case, the network appeared to learn faster from the training set, and the performance on the validation seemed to improve at the same rate, although it always remained about one percentage point below the other.

Training loss	**Training accuracy**	**Validation loss**	**Validation accuracy**
0.1064	0.9612	0.3400	0.8715




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_17 (InputLayer)        [(None, 224, 224, 3)]     0

tf.math.truediv_12 (TFOpLam  (None, 224, 224, 3)       0
bda)

tf.math.subtract_12 (TFOpLa  (None, 224, 224, 3)       0
mbda)

mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)        2257984
ional)

global_average_pooling2d_3   (None, 1280)              0
(GlobalAveragePooling2D)

dropout_5 (Dropout)          (None, 1280)              0

dense_16 (Dense)             (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,985,793
Non-trainable params: 273,472
```

Figure 30. Learning curves and summary of the network

By evaluating the performance of the model on the test set, we could see an improvement. In fact, this model achieves an F1-score of 0.8717 and an AUC of 0.93. In any case, despite the improvement, the network performs worse than our CNN from scratch and therefore we cannot be satisfied with these results.

Precision		**Recall**		**Weighted F1-Score**
Normal	*Cancer*	*Normal*	*Cancer*	
0.7823	0.9179	0.8362	0.8873	0.8717

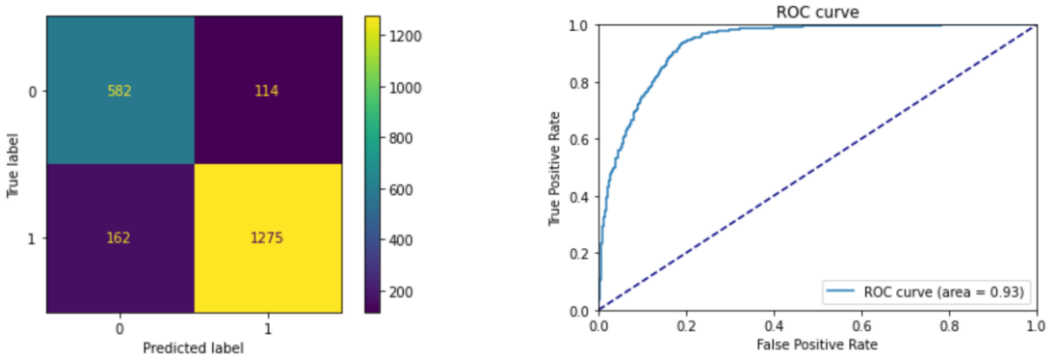


Figure 31. Confusion matrix and ROC curve

3.3.3.2 Removing Last Block

Since with the other pre-trained networks it had always brought benefits in terms of performance, once again we tried to remove the last block and re-run the feature extraction and finetuning experiments.

However, unfortunately with MobileNetV2 we were not able to achieve satisfactory results and the performance remained the same as in the previous case, if not worse. At this point, since we had already obtained very good performance from the other networks and this one did not seem to behave as we expected, we decided to discontinue the experiments on MobileNetV2 and to try another way: ensembling.

3.4. Ensemble

Model ensembling is a very powerful technique that typically achieves the best possible results. As a matter of fact, many of the solutions employed in the state of the art and by the participants in the challenge involved the use of ensembles. For this reason, we considered combining the predictions of our best models.

3.4.1. Error Analysis

So far, what we have been able to see is that most of the models perform very well on the class of interest, but less well on the normal cell class. This is obviously more desirable than the opposite case, but still ideally, we would like a classifier that can correctly classify both classes. The goal thus is to find a model with an increased F1-score while keeping the recall on the class of interest as high as possible.

In order to evaluate the convenience or not of resorting to ensemble solutions, an analysis of the errors committed by the models with the best performances has been carried out. Since the diversity of the selected set of classifiers is key to the proper functioning of ensembling, it was important to analyze the errors in order to understand whether they were common to all the classifiers or whether each classifier had different predictive power despite having similar performance.

The first thing that could be noticed is that 41 images in the test set are misclassified by all four classifiers. Of these images, a portion of which is shown in Figure 32, only 9 represent cancerous cells, while the remainder represent healthy ones. Repeated errors on the healthy cell class could likely be due to poor representation for that class within the training set. In fact, the number of available images may not have been sufficient to generalize the characteristics of normal white blood cells. Making further considerations is not easy because, as already mentioned, the task of distinguishing the two types of cells is extremely challenging even for experts in the field. In any case, to the untrained eye, the misclassified images appear to have in common the presence of more black background and relatively more irregular shapes when compared to the rest of the dataset.

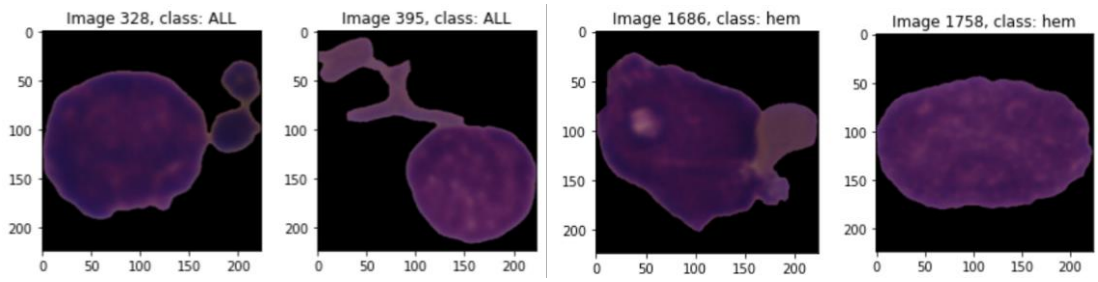


Figure 32. Some images misclassified by all classifiers

Another thing we could notice is that most of the errors are made by only one classifier at a time. This means that, for 187 of the misclassified images, where one classifier is wrong, the other three manage to give the correct prediction. This led us to believe that we might benefit from an ensemble solution. In fact, as can be seen in Figure 33, the images misclassified by all or most classifiers are less than half of the total misclassified images.

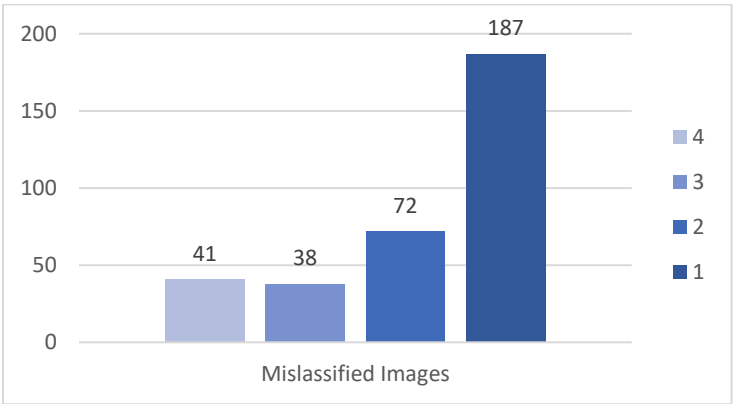


Figure 33. Misclassified images by number of classifiers that misclassify them

Furthermore, by analyzing what the classifiers got wrong and what they got right, we were able to infer something interesting. Although the ResNet50: version 1 model was the second best in terms of weighted F1-score and the first best in terms of recall on the class of interest, it does not seem to have greater predictive power than the other models. Specifically, while the other models have 12 or 13 images that only they are able to classify correctly, ResNet50: version 1 classifies only one image that no one else can classify. This means that every image that this model classifies correctly, apart from one, is correctly classified by at least one other model.

3.4.2. Average Model

Since we have already trained all models individually and can retrieve their predictions on the test set, the easiest way to pool their predictions is to average them at inference time. Having four different models, this results in

*ensemble_prediction = 0.25 * (prediction_a + prediction_b + prediction_c + prediction_d)*

With just this simple operation, the resulting ensemble model manages to outperform the four models individually, improving the F1-score to 0.9503 and the recall on the class of interest to 0.9805. Also, the AUC for this model is 0.98.

Model	**Precision**		**Recall**		**Weighted F1-Score**
	Normal	*Cancer*	*Normal*	*Cancer*	
CNN from scratch	0.8754	0.9230	0.8376	0.9422	0.9076
VGG16	0.8522	0.9400	0.8779	0.9262	0.9108
ResNet50: version 1	0.9512	0.9386	0.8678	0.9784	0.9416
ResNet50: version 2	0.9214	**0.9564**	**0.9095**	0.9624	0.9451
Ensemble: average model	**0.9567**	**0.9482**	**0.8894**	**0.9805**	**0.9503**

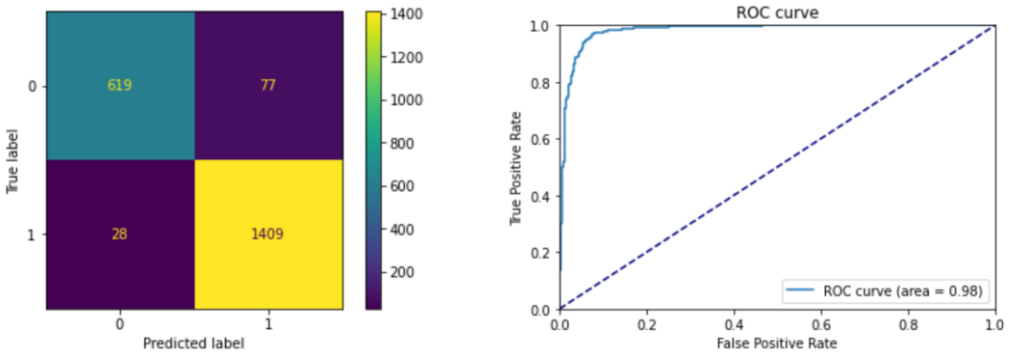


Figure 34. Confusion matrix and ROC curve

3.4.3. Weighted Average Model

All classifiers are more or less equally good but, in order to make the most of the ensemble technique, it is smarter to take a weighted average of the predictions so that the best classifiers have higher weights than the worst classifiers. In general, a good set of ensembling weights should be learned from the validation data. There are many possible techniques, but we decided to implement a Genetic Algorithm that can find an optimal solution to the problem.

3.4.3.1 Genetic Algorithm

We defined the problem of finding the best set of weights as follows:

- Each chromosome is composed of four genes (real encoding) that represent the weights associated with the four models that make up the ensemble.
- The initial population of candidate solutions is randomly generated, ensuring that the sum of the different genes for each chromosome is equal to 1 (constraint imposed by being weights).
- As fitness function, since the validation set is unbalanced, weighted F-measure and not the accuracy is used. The goal is then to maximize this function.
- As the technique for selecting potential parents, a k-tournament selection is used.

- The average crossover is used as crossover operator. Normalization is added in order to comply with the constraint on the sum of the weights.
- For mutation, a single gene is chosen at random and is replaced with a randomly generated real number in the range [0,1]. Subsequently, normalization results in an all-positions mutation.
- The number of individuals in the population is always the same for each generation.

The parameters of the genetic algorithm, namely population size, number of generations, and crossover and mutation probabilities were chosen experimentally. Furthermore, the entire algorithm was run multiple times with different initial populations and the best solution was considered, i.e., the one that resulted in the best F1-score on the validation set.

3.4.3.2 Ensemble Results

The weights obtained through the genetic algorithm were used to calculate the weighted average of the predictions of the individual classifiers. As expected, the resulting ensemble obtains a weighted F1-score of 0.9552, which is better than the previous one. This improvement in the weighted F1-score is due to a marked improvement in recall for the negative class, at the expense of recall for the positive class, which nevertheless remains high.

Model	**Precision**		**Recall**		**Weighted F1-Score**
	Normal	*Cancer*	*Normal*	*Cancer*	
CNN from scratch	0.8754	0.9230	0.8376	0.9422	0.9076
VGG16	0.8522	0.9400	0.8779	0.9262	0.9108
ResNet50: version 1	0.9512	0.9386	0.8678	0.9784	0.9416
ResNet50: version 2	0.9214	0.9564	0.9095	0.9624	0.9451
Ensemble: average model	**0.9567**	0.9482	0.8894	**0.9805**	0.9503
Ensemble: weighted average model	**0.9478**	**0.9590**	**0.9138**	**0.9756**	**0.9552**

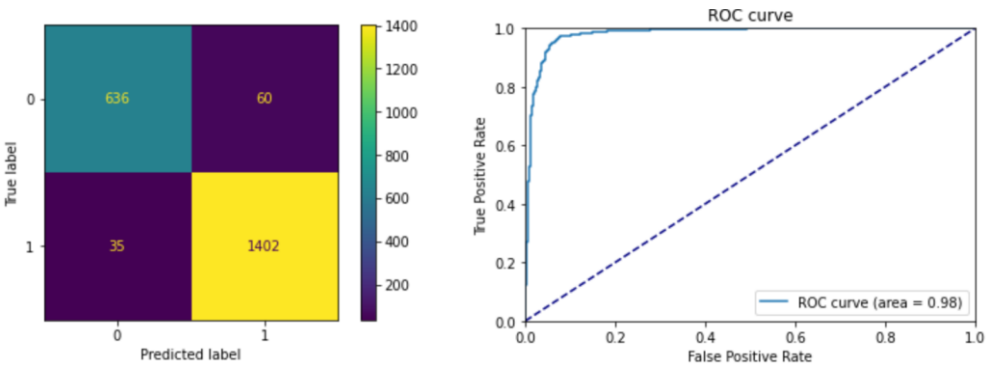


Figure 35. Confusion matrix and ROC curve

4. EXPLAINABILITY

Now we will try to understand how out networks classify the images. There are a lot of possible techniques, we decided to focus on Intermediate Activations and Heatmaps of the class activations.

4.1. Intermediate activations

Visualizing intermediate activations consists in displaying the feature maps that are outputted by the convolutional and pooling layers in the network, given a specific input.

To see the differences between the models we will consider the same image.

4.1.1. CNN from scratch

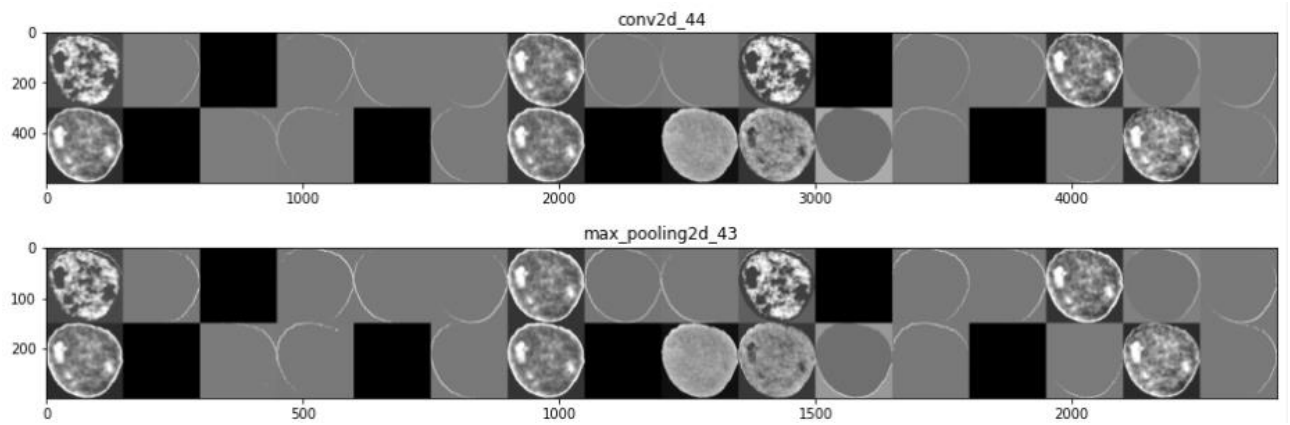


Figure 36. Intermediate activations CNN from scratch

In the first convolutional layer the network focuses primarily on the edges of the cell. In fact, most filters analyze the edges. Some activations are completely black, this is mainly due to the use of the ReLU, which brings the output to 0 in the case of negative values. One filter also focuses on the part of the background outside the cell, giving meaning to the relative size of the cell. Finally, there are also some filters that analyze the inside of the cell.

4.1.2. ResNet-50

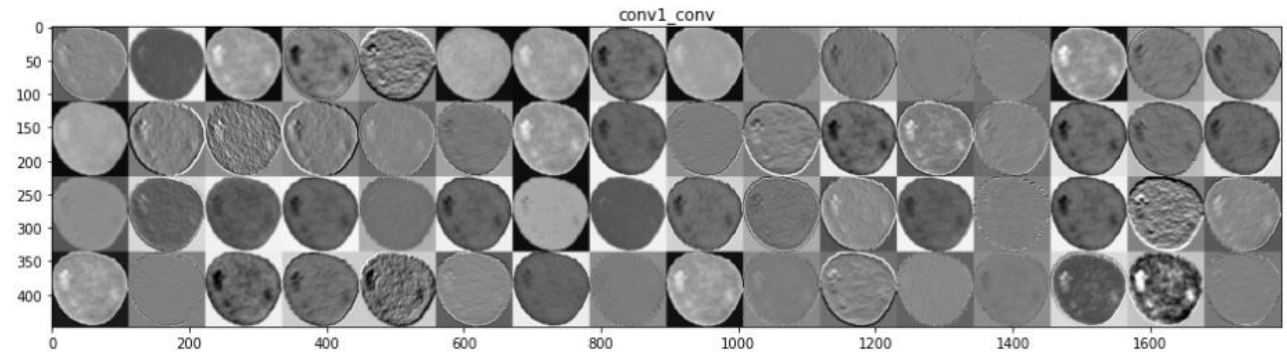


Figure 37. Intermediate activations ResNet-50

The intermediate activations are remarkably similar in the two models of ResNet, for simplicity we analyze only those of the first model.

In the first convolutional layer the network focuses a lot on the background outside the cell, giving much importance to the relative size of the cell. In this case there are no filters that analyze only the edges, but they are considered together with internal details of the cells.

4.1.3. VGG16

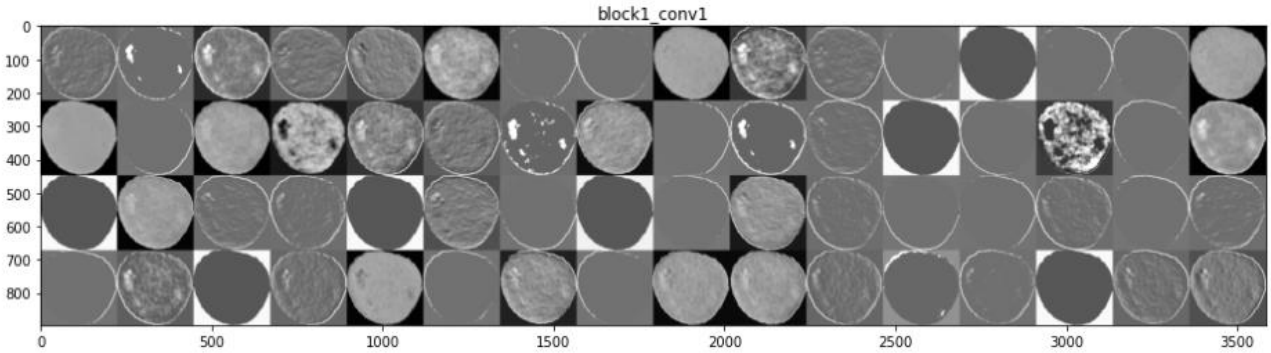


Figure 38. Intermediate activations VGG16

In this case there are both filters dedicated only to edge analysis and filters that focus on the background outside of the cell. This appears to be an intermediate behavior to that seen for the other networks. Of course, there are also filters that consider internal details already in the first convolutional layer.

4.2. Heatmaps of Class Activation

This technique is useful to understand which parts of a given image led a convnet to its final classification decision. We will briefly analyze the behavior of the networks when they receive as input an image of a cancer cell and an image of a healthy cell.

4.2.1. Cancer cell

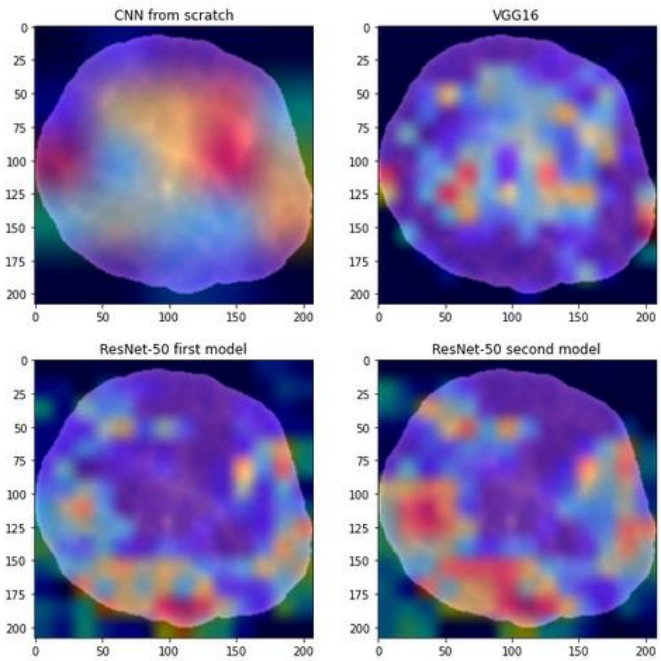


Figure 39. Heatmaps of cancer cell

First, it is important to note that all networks focus on details within the cell. This is an extremely positive fact, because they mainly distinguish the cell based on its internal features and not because of the background. There are not too many

differences between the two ResNet models, although the latter model tends to focus on more detail than the former. All the networks focus on different details, which is why the ensemble network gets better results.

4.2.2. Normal cell

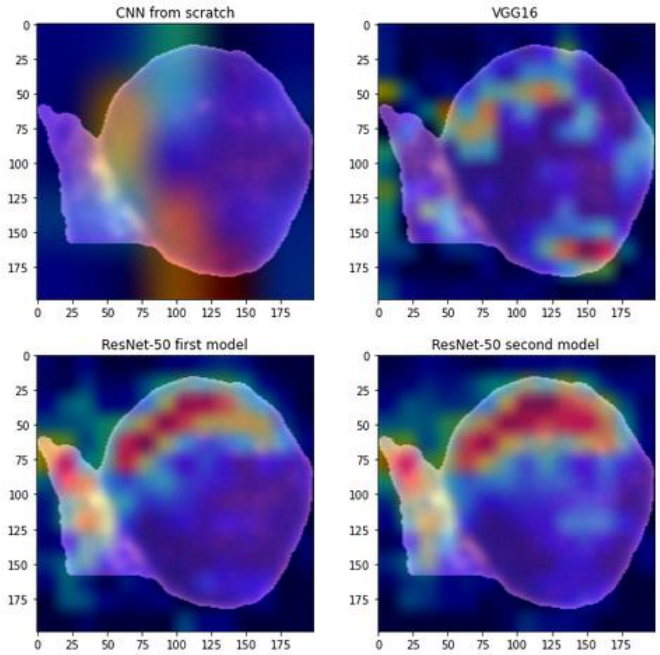


Figure 40. Heatmaps of normal cell

Again, we note how the networks focus on different details, although there are fewer differences between ResNet networks. In this case, some networks focus more on external details, and this may be due to the smaller amount of negative class images available for training. With a larger dataset, and thus greater computational capabilities to exploit it, improvements in negative image classification can be achieved.

5. CONCLUSIONS

In this work we tested both from scratch and pre-trained networks for the classification of malignant lymphoblasts cells. Then we proposed an ensemble solution able to combine the classification power of the best networks, optimizing the weights with a genetic algorithm.

The results obtained are encouraging and show that it is possible to correctly classify almost all malignant cells, with a precision of 95.90% and a recall of 97.56%. Overall, our ensemble achieves an F1 weighted score of the 95.52%. Since we did not have high computational capabilities we could use only a part of the dataset, and this could lead to an unsatisfactory accuracy in the analysis of malignant cells of rare shape that are not present in our dataset. So, despite the promising results, cell classification problem is far from solved.

There is still room for improvement, such as the use of hyperparameter optimization to further optimize our networks, which again for computational reasons was not possible.

Finally, it would also be interesting to work directly on the raw images, while in the competition dataset they were already preprocessed.

6. REFERENCES

[1] National Comprehensive Cancer Network, "Acute Lymphoblastic Leukemia," [Online]. Available: https://www.nccn.org/guidelines/guidelines-detail?category=1&id=1410. [Accessed 20 December 2021].

[2] CodaLab, "Classification of Normal vs Malignant Cells in B-ALL White Blood Cancer Microscopic Image: ISBI 2019," [Online]. Available: https://competitions.codalab.org/competitions/20395. [Accessed 20 December 2021].

[3] A. Gupta and R. Gupta, "ALL Challenge dataset of ISBI 2019," 2019. [Online]. Available: https://doi.org/10.7937/tcia.2019.dc64i46r. [Accessed 20 December 2021].

[4] A. Gupta, R. Duggal, S. Gehlot, R. Gupta, A. Mangal, L. Kumar, N. Thakkar and D. Satpathy, "GCTI-SN: Geometry-inspired chemical and tissue invariant stain normalization of microscopic medical images," *Medical Image Analysis,* vol. 65, 2020.

[5] R. Gupta, P. Mallick, R. Duggal, A. Gupta and O. Sharma, "Stain Color Normalization and Segmentation of Plasma Cells in Microscopic Images as a Prelude to Development of Computer Assisted Automated Disease Diagnostic Tool in Multiple Myeloma," *16th International Myeloma Workshop New Delhi,* vol. 17, no. 1, 2018.

[6] R. Duggal, A. Gupta, R. Gupta, M. Wadhwa and C. Ahuja, "Overlapping cell nuclei segmentation in microscopic images using deep belief networks," *ICVGIP '16: Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing,* pp. 1-8, December 2016.

[7] E. Verma and V. Singh, "ISBI Challenge 2019: Convolution Neural," in *ISBI 2019 C-NMC Challenge: Classification in Cancer Cell Imaging*, Springer, 2019, pp. 131-139.

[8] A. Honnalgere and G. Nayak, "Classification of Normal Versus Malignant Cells in B-ALL White Blood Cancer Microscopic Images," in *ISBI 2019 C-NMC Challenge: Classification in Cancer Cell Imaging*, Springer, 2019, pp. 1-12.

[9] C. Marzahl, M. Aubreville, J. Voigt and A. Maier, "Classification of Leukemic B-Lymphoblast Cells from Blood Smear Microscopic Images with an Attention-Based Deep Learning Method and Advanced Augmentation Techniques," in *ISBI 2019 C-NMC Challenge: Classification in Cancer Cell Imaging*, Springer, 2019, pp. 13-21.

[10] F. Xiao, R. Kuang, Z. Ou and B. Xiong, "DeepMEN: Multi-model Ensemble Network for B-Lymphoblast Cell Classification," in *ISBI 2019 C-NMC Challenge: Classification in Cancer Cell Imaging*, Springer, 2019, pp. 83-93.

[11] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.

[12] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2015.

[13] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," 2019.