Myocardial infarction prediction

SEMESTER PROJECT

OMAR RAITA

PROFESSOR: FROSSARD PASCAL

SUPERVISORS: Dorina Thanou, Ortal Senouf

June 11, 2021

Contents

List of Figures

1 Introduction

A myocardial infarction is a cardiovascular disease that occurs when blood flow decreases or stops to a part of the heart. The MI causes damage to the heart muscle which may lead to an irregular heartbeat, a heart failure or even a cardiac arrest [1]. An MI is usually caused by coronary artery diseases like stenosis (Fig.1) and can be diagnosed in different ways such as electrocardiograms, blood tests or coronary angiography.

The objective of this work is to use machine learning and deep learning techniques to predict, at the stenosis level, whether it will cause an MI or not. And ultimately, be able to help cardiologists with the diagnosis of MIs at an early stage. In order to do so, some challenges will need to be addressed such as the limited and imbalanced dataset. Such a task may also require including prior knowledge from the medical field as MIs can be caused by other systemic factors such as cellular signaling, fluid hemodynamics and vascular wall histology.
This report outlines the different steps that were taken to construct a machine learning pipeline to predict MIs, from the data preparation to the training and evaluation methods. It also presents different approaches that were considered in order to improve our model's performance and robustness. In the last section of the report, we will provide performance results on the testing set as well as a detailed comparison between the most promising models.

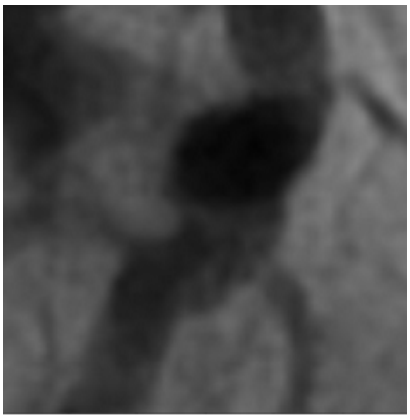

Figure 1: X-ray coronary angiography of a stenosis.

2 Literature Review

Most of deep learning works that were conducted on coronary angiography were focusing on stenosis detection or vessels segmentation tasks. Some works include more advanced topcis such as the one from the Automatic analysis for coronary angiography paper [2]. In which the authors present a deep learning system (Deep Discern) that combines the results of two networks to provide cardiologists with a "coronary diagnosis map" that includes different information about the coronary angiography. The first network is a conditional generative adversarial network that is used to identify different types of coronary artery segments within the angiography. For this task, 13 373 angiograms labelled with 20 different types coronary artery segments are used. The second network is a convolutional DNN that was trained on 7 239 labelled angiograms, to recognize different lesion characteristics such as the lesion location, its diameter, calcification and other characteristics. The model achieved 98.4% accuracy for the segments recognition and 0.82 F1-score for the lesion morphology detection task.

Another work was done to automatically recognise and localize stenosis from coronary angiography clips [3]. In this paper, the authors first used a key frame extraction method from the clips, which consists of extracting the frame with the higher number of white pixels, after enhancing the contrast and applying a Frangi-filter [4]. Next, a GoogleNet Inception-v3 network is used to classify (normal or abnormal) 45 135 frames, obtained by 25 times data augmentation. Finally, they use a self attention mechanism to detect the stenosis location. The training achieved a frame-wise accuracy of 0.934

Regarding the vessels segmentation task, an interesting paper proposes "a robust method for major vessels segmentation" [5], where the authors train a U-Net network on 3302 angiograms of diseased major vessels from 2042 patients, to achieve real time segmentation with minimal image processing. A comparison between different backbones is provided as well. The authors were able to reach an average F1-score higher than 0.8.

3 Data preparation

3.1 Dataset

The dataset was obtained from a clinical study that consists of 83 patients from CHUV. Four images corresponding to four different views were taken from each patient. On top of that, each image was manually annotated with green and red dots by cardiologists from CHUV, which makes a total number of 374 raw and 374 labelled images of 1014x1014 pixels.

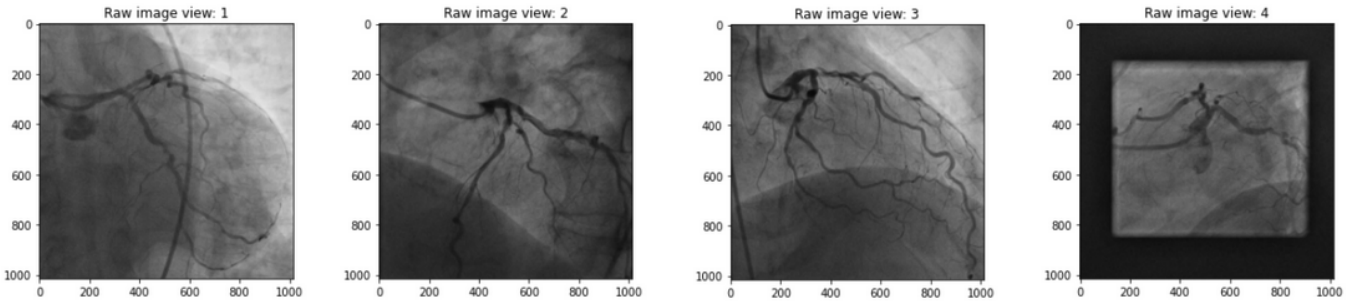

Figure 2: Four different views of a Coronary angiography taken from the same patient.

In the annotated images (Fig. 3) , a red dot on the stenosis means that the latter caused a myocardial infarction after a period of 1 to 5 years, while a green dot means that no MIs were caused by the stenosis.

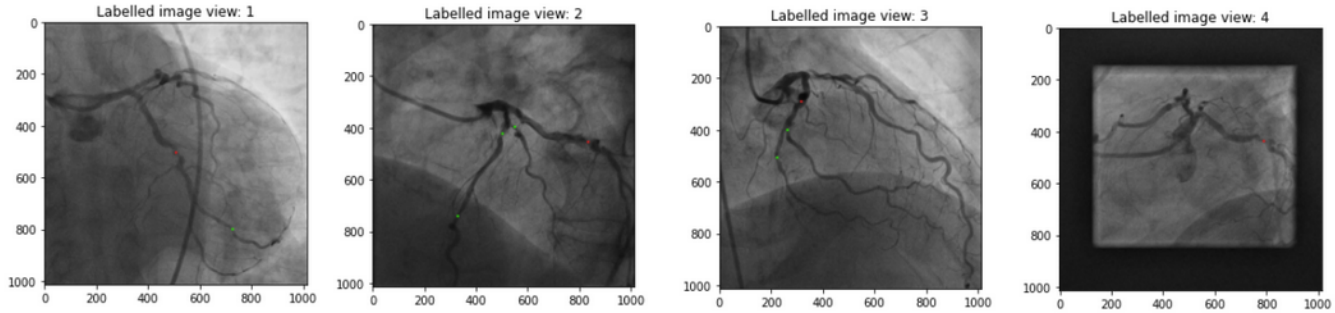

Figure 3: Annotated views of a Coronary angiography taken from the same patient.

3.2 Labels detection

The training dataset was constructed by first reading the labels' colours on the annotated images and then extracting the corresponding patches around the stenosis region from the raw images. A label is assigned to each patch depending on the detected colour (1 for red and 0 for green). The colours are detected by iterating over all the pixels of the labelled image and comparing the Red and Green channels' values to a certain threshold. Once a red or green colour is detected (Fig. 4), its position and label are saved and passed to another function that will use the same position to crop patches from the raw images. Note that the same dot can not be detected twice as an area of 30x30 around the detected pixel is skipped in the next iterations.

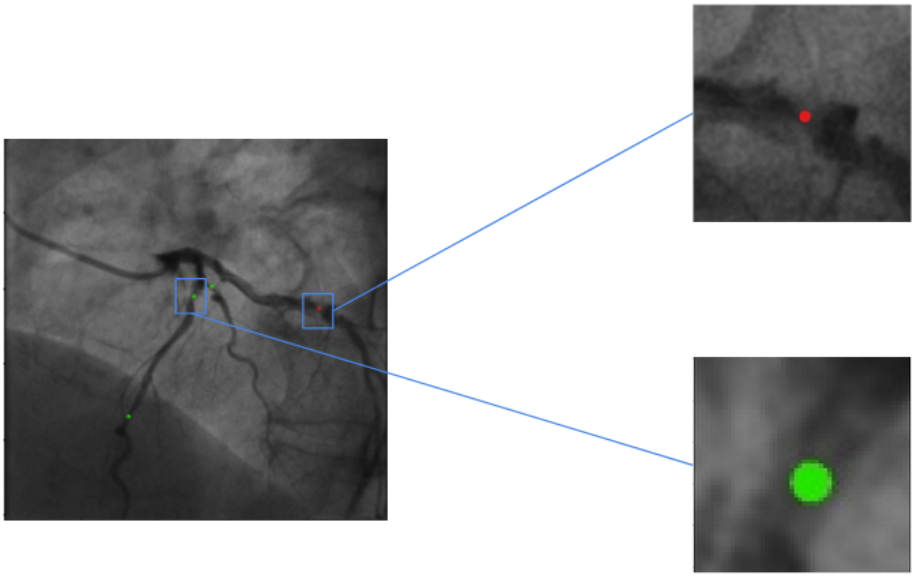

Figure 4: Annotated patches extraction from an annotated CA.

3.3 Patches creation

For a given label location, the distance to the image borders and all other labels in the same image is computed and used to crop the largest possible square around the detected pixel (Fig. 5). If no labels or borders are near the given location, the cropped patch will be of size 224x224.

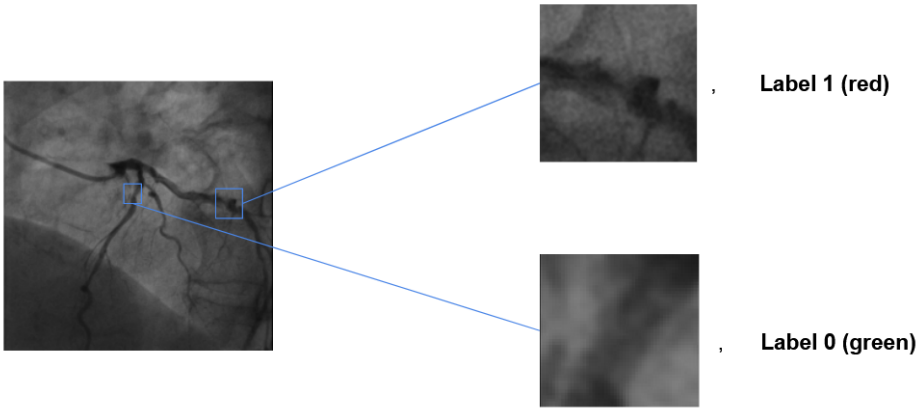

Figure 5: Patches extraction from the raw CA.

A Dataset splits

After having created the patches and the labels, the dataset was split into a training, validation and test sets. Table 1 gives approximate values of the proportion of the data in each set.

Dataset	Total number of patches	Percentage
Train set	522	70%
Validation set	142	20%
Test set	75	10%

Table 1: Dataset splits.

3.4 Data transformation

A sequence of transformations from the Albumentations library [6] was used in the data augmentation as well as the training phase:

- Median Blur: Blurs the input image with a random-sized kernel.
- Rotation: Rotates the image to 90°, zero or more times with a random probability.
- ShiftScaleRotate: Rotates and scales the image with a random probability.
- Resize: Resizes all the patches to the same size (224x224).

4 Training

The training is composed of two phases. A first phase, where we train the Pytorch implementation of the *resnet*18 network on the training set by minimizing the Cross Entropy Loss with a stochastic gradient descent optimizer. In the second phase, we perform simple forward passes on 20 samples batches from the validation set to assess the network's performance.
The performance metrics that are used in our training are the following:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Specificity = \frac{TN}{TN + FP} \tag{2}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{3}$$

$$F1\_measure = \frac{TP}{TP + 0.5(FP + FN)} \tag{4}$$

The first training is performed on a dataset that is composed of 150 Red Patch and 370 Green patch, without any data augmentation. As a first experiment, we compare the performance of two *resnet*18 models. The *Scratch* model is a randomly initialized *resnet*18 network while the *Pretrained* model uses weights that are pre-trained on ImageNet.

The first results (Fig. 6) show that both models are overfitting the green class ($non - culprit$) as both sensitivities are very low. In fact, the validation accuracy results are not reliable since the class distribution is highly imbalanced.

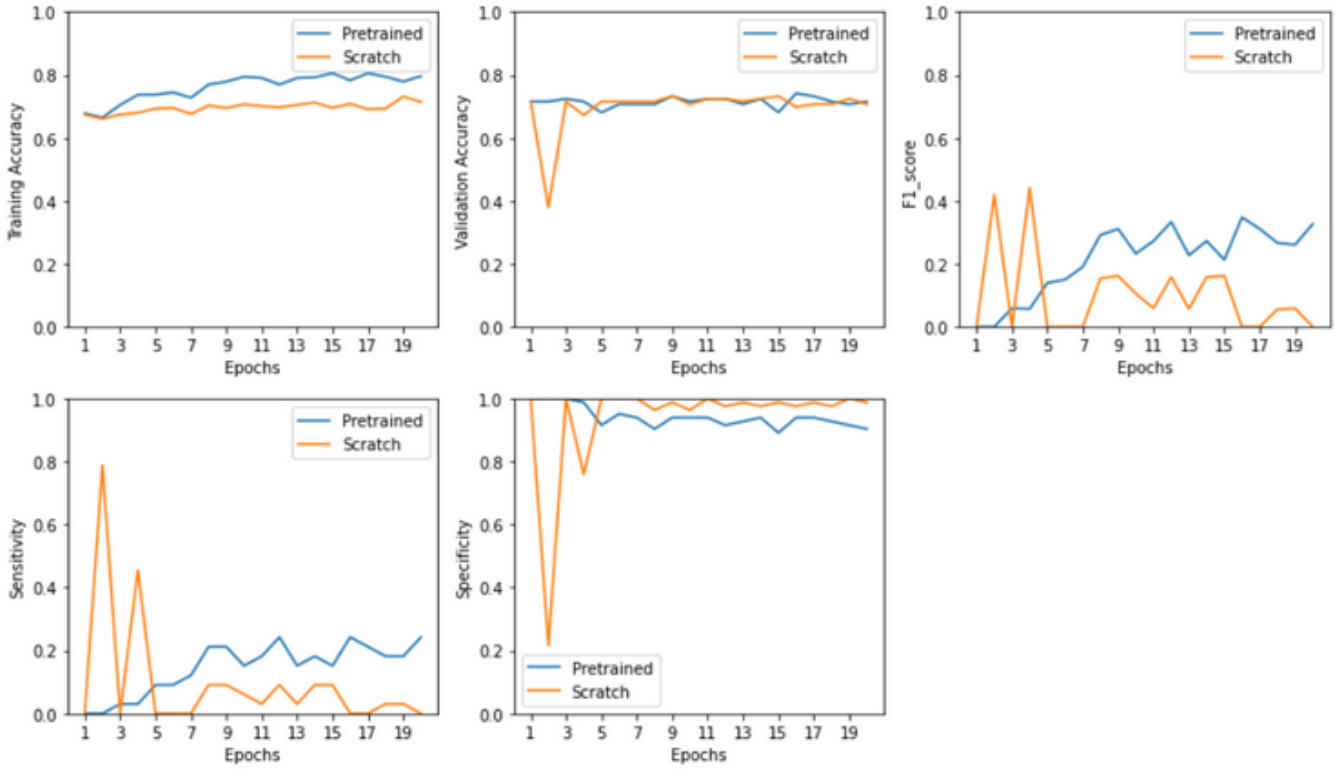

Figure 6: Baseline results for the pretrained and scratch model.

5 Data augmentation techniques

In this section, we will present the most relevant data augmentation techniques that were investigated in order to tackle the classes distribution problem.

5.1 Weighted dataloader

The weighted data loader is a technique that allows to load batches containing balanced samples between the two classes. To do so, we should first create a weighted matrix that can be defined as follows:

$$Weights = \begin{bmatrix} \frac{N}{C_0} & \frac{N}{C_1} \end{bmatrix} = \begin{bmatrix} 1.40 & 3.47 \end{bmatrix}$$

Where N is the total number of patches from both classes. C_1 and C_2 are, respectively, the number of patches from class 1 and class 2.

The weighted matrix is passed as an argument to the torch *WeightedRandomSampler*, which can

be used as a Dataloader sampler in the data loading phase.

This sampler multiplies the samples loading probabilities by their corresponding weights, which allows to load samples with balanced probabilities as showed in Figure 7.

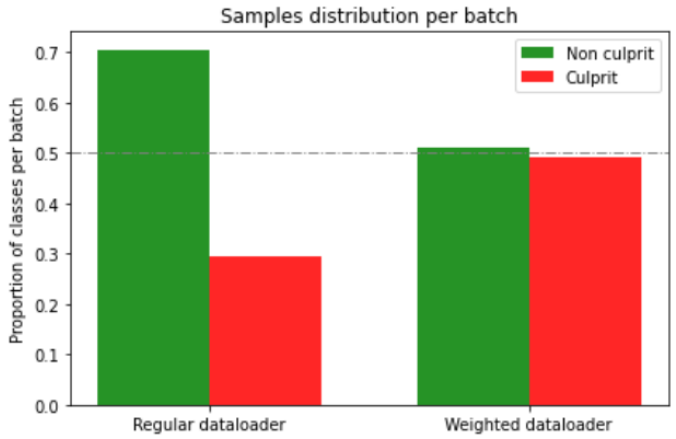

Figure 7: Weighted Dataloader effect on the samples distribution.

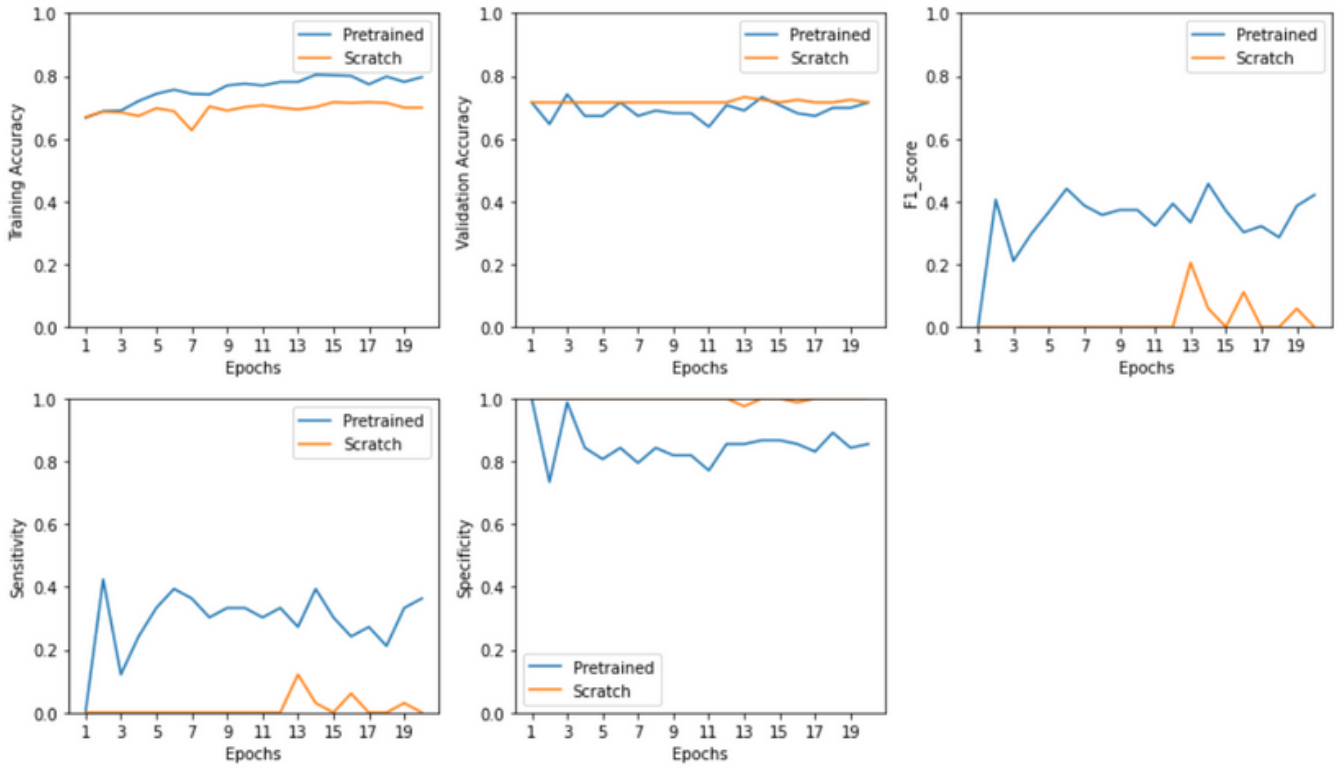

Figure 8: Performance results for the pretrained and scratch models, using the data loader

One can observe an improvement of the *Pretrained* model in terms of sensitivity and therefore F_1 score (Fig. 8). The *Scratch* model results show that it has completely overfitted the

$non-culprit$ class by consistently predicting 0. This could mean that the *Scratch* model was not learning meaningful features in the previous experiment for the $Culprit$ class.

5.2 Data balancing

The *Pretrained* model has indeed showed some improvements in terms of sensitivity when we used a balanced dataloader, but the bias towards predicting $non-culprit$ was still observable. We therefore tried new experiments such as data balancing by replicating the red patches until both classes are balanced. The replications were combined with data transformations as described in section 3.4. Note that both the training and validation sets were balanced while the test set remained unchanged.

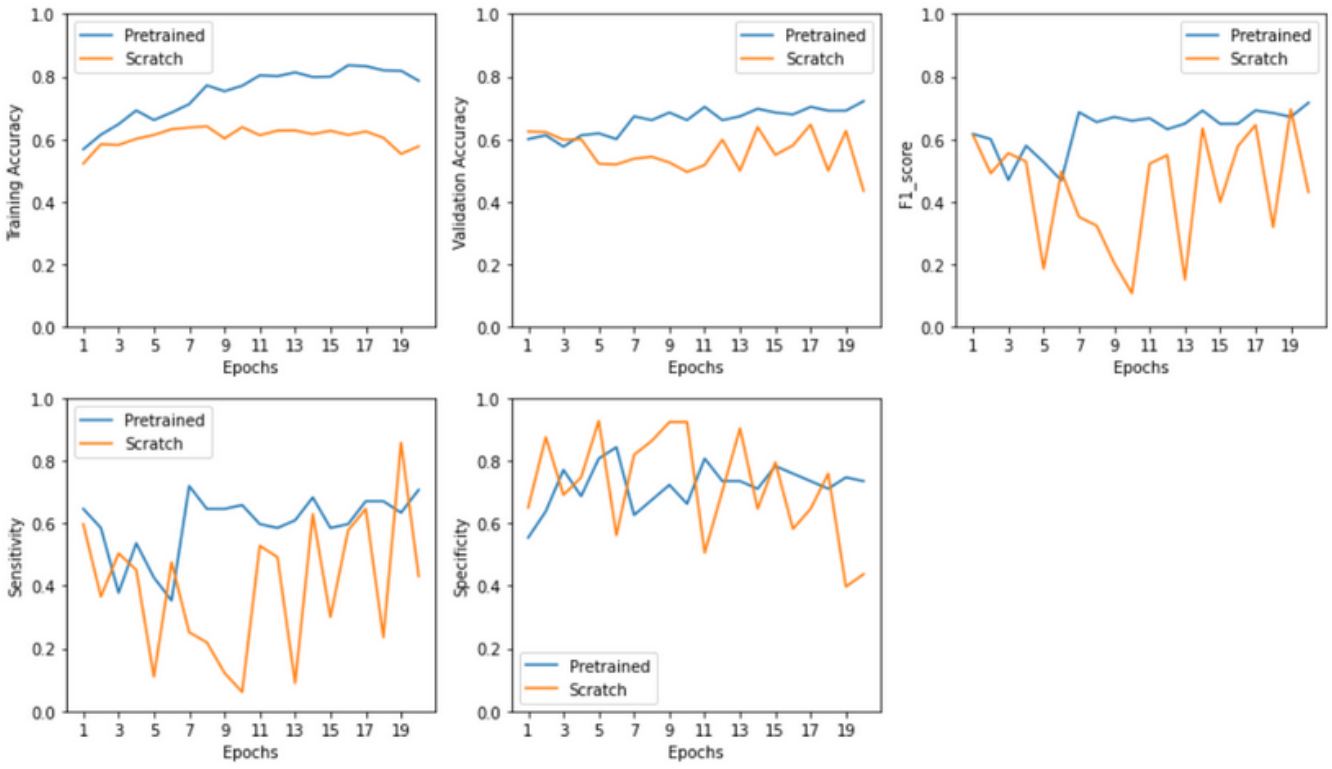

Figure 9: Performance results for the pretrained and scratch models after data balancing

Notable improvements can be seen in both models after balancing the data (Fig. 9). The *Scratch* model can finally yield meaningful results but does not seem to learn significant features as the validation accuracy is just above 50%. Note that, for this experiment, it is possible to rely on

the validation accuracy as a performance metric since both classes are balanced (370 patch per class). The *Pretrained* model is able to achieve a valiation accuracy of 70% with an *F*1_*score* of 0.7.

5.3 Augmentation by different multiplication factors

In addition to the data balancing, other augmentations were added by applying the same replication procedure as in the previous experiment, but this time on both classes. In this experiment, we have tried multiple augmentation factors (up to $x4$). The results that are retained are the ones for the $2x$ augmentation factor, meaning that we have a total number of 740 patch for each class.

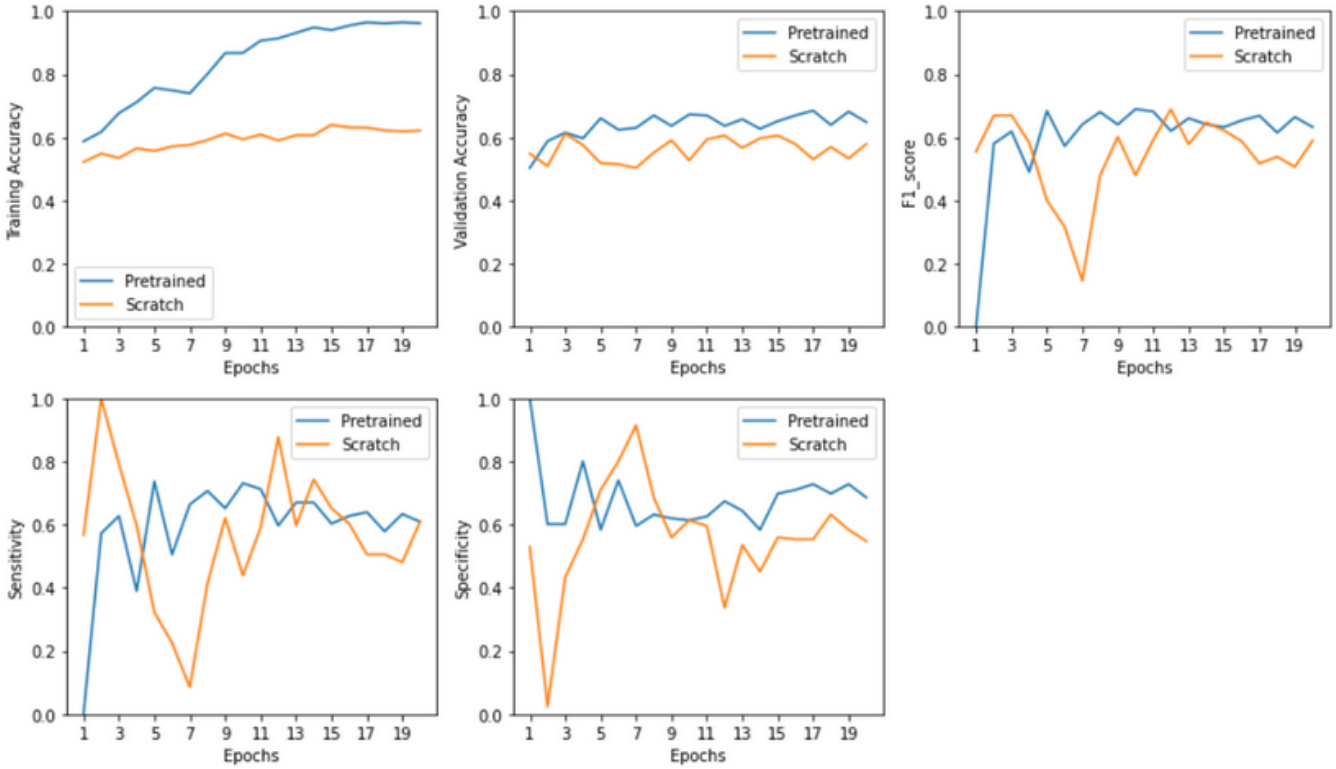

Figure 10: Performance results for the pretrained and scratch models after balancing and multiplying the whole data by 2.

The *Pretrained* model starts overfitting both classes when we use a multiplication factor greater than $x2$. This can be seen in the training accuracy plot that almost achieves 100% after 20 training epochs. On the other hand, the scratch model has showed some improvements (60% validation

accuracy), in comparison with the previous experiments where its predictions are almost equivalent to random guesses.

5.4 Cross validation

In order to select the best set of hyperparameters, a 5-fold cross validation was performed on the imbalanced training set, for different parameters.
We couldn't run the cross validation on the balanced set as the 20% validation fold may contain replicated patches that are also contained in the 80% training fold. We use the $F1score$ as the hyperparameters selection metric since the data is imbalanced. We manually tried different hyperparameters to detect the most influent ones. We found out that both the learning rate and the weight decay yield non-negligible changes in the validation results. The figures below show the cross validation results for both parameters.

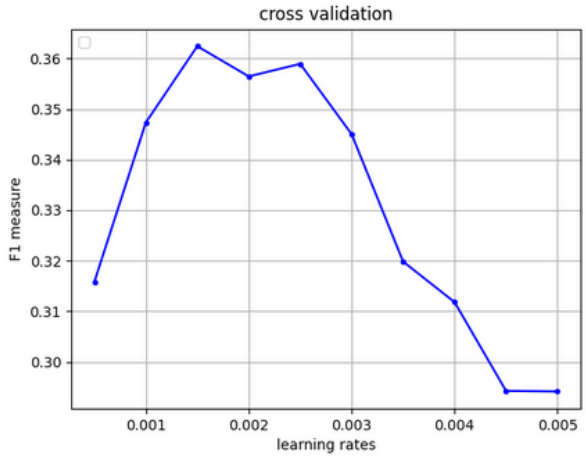

Figure 11: Cross validation over the learning rate with a fixed weight decay.

Based on the 2D-cross validation results (Fig. 13), the final set of relevant hyperparameters that is chosen is the following:

- learning rate = 0.002
- weight decay = 0.0227

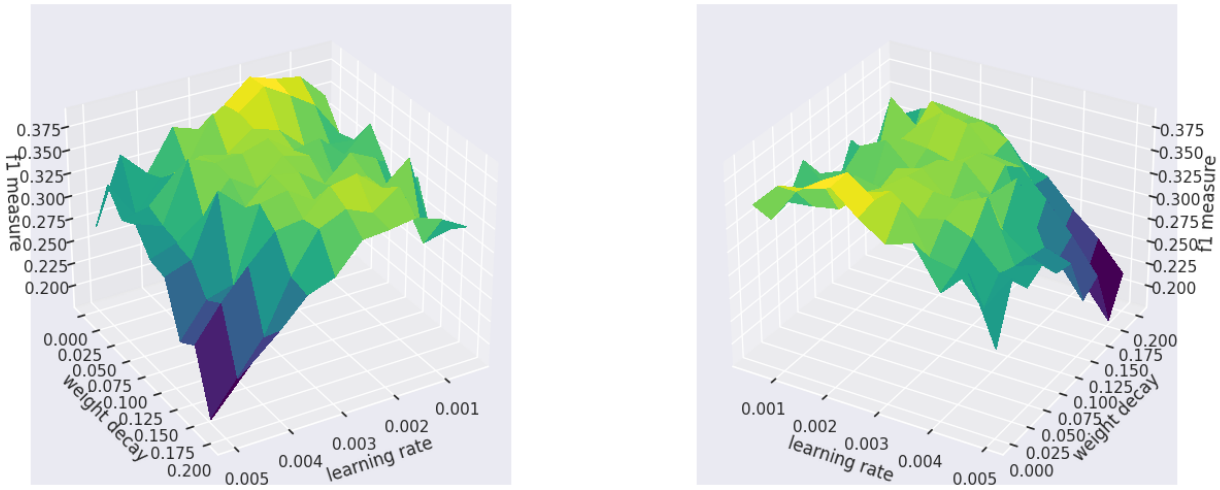

Figure 12: Cross validation over the learning rate and weight decay.

As it can be seen in Figure 13, for a given learning rate (e.g 0.0015), the f1 score may change from 0.32 to 0.38, depending on the weight decay value. Therefore, a 2D cross validation was necessary to find the best set (learning rate, weight decay). The 3d visualizations were plotted to show the smoothness of the results. This means that it is possible to choose any value around the target value and that there is no need to run a tighter grid search.

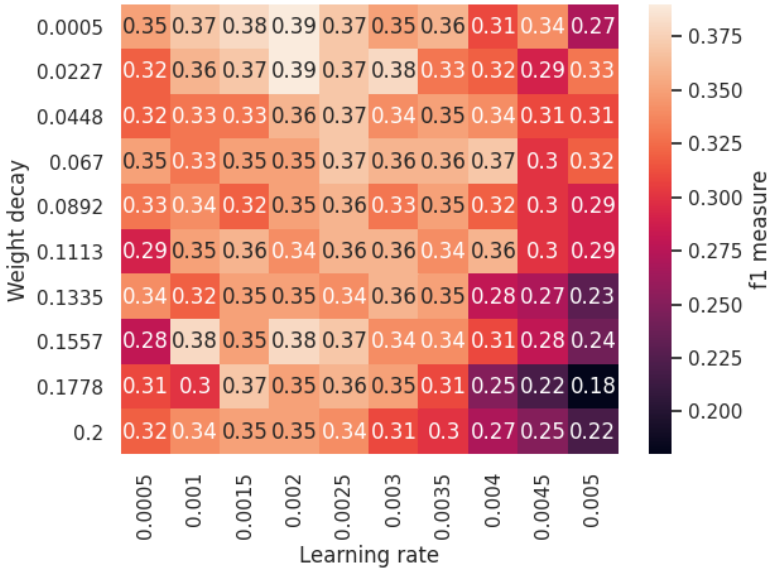

Figure 13: f1-measure heatmap over the learning rate and weight decay.

5.5 Data augmentation technique and model selection

The selected model is the *Pretained* model as it has completely outperformed the *Scratch* model in all the experiments. We train this model on the balanced data set, with no extra- augmentations, using the set of hyperparameters that were found in section 5.4.

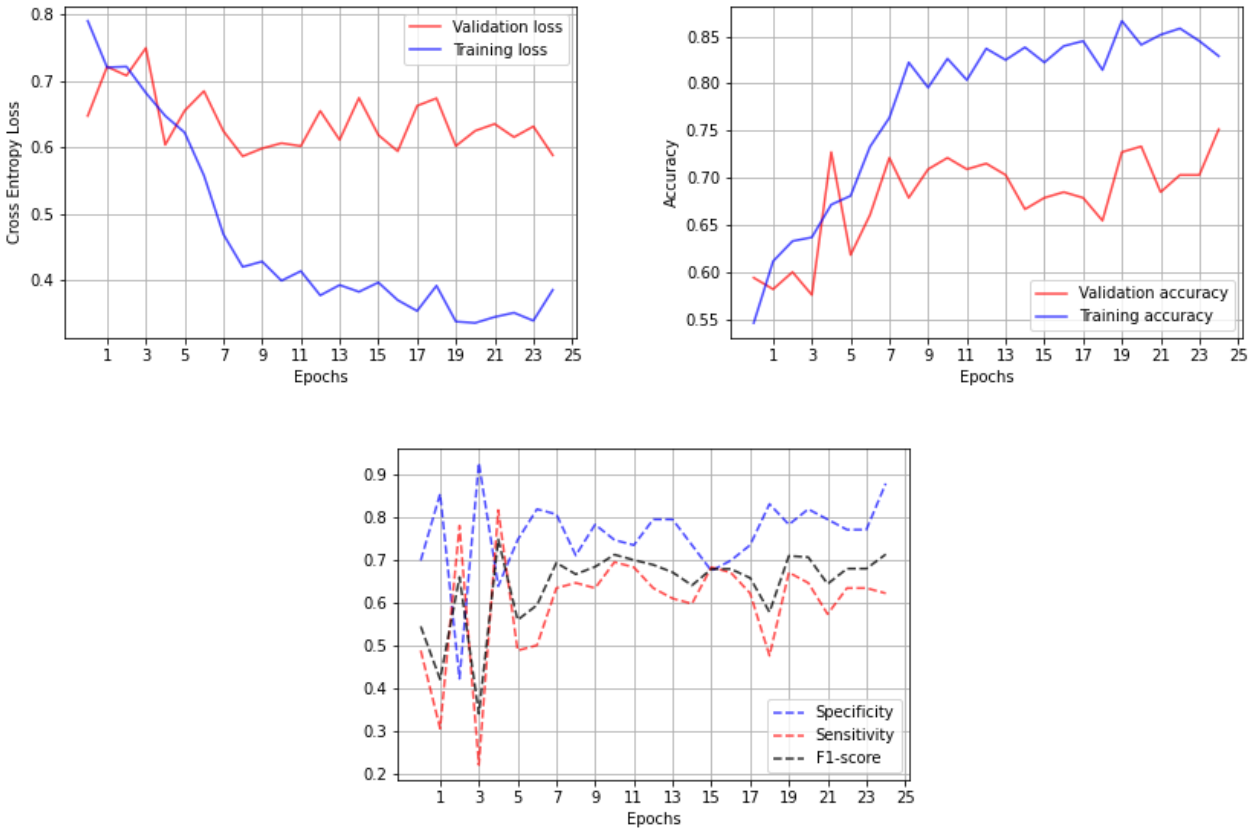

Figure 14: Final results of the pre-trained model

After selecting the best hyperparameters, the validation accuracy of the pre-trained model has improved from 70% to 73%, on average across different runs. Sometimes reaching even higher values (e.g. 75%) as it can be seen in Figure 14.

6 Feature extraction

In the second part of the project, we have explored different ways to help the model focus on important features within the patches.

6.1 Frangi filter

The Frangi filter is a vessel enhancement technique that is used to detect tubular-like structures in an image. This filter was used in many deep learning tasks such as the Retinal vessel segmentation [7] or as pre-processing method on angiogram clips for key frames detection, as described in the aforementioned paper [3].

The frangi filter is computed from the eigen values of the Hessian matrix of an Image. The hessian matrix can be obtained by first smoothing the image with a gaussian filter g_σ and then computing the second derivative of the smoothed image.
Alternatively, it is possible to compute the Hessian H_σ by directly convolving the initial image I with a gaussian kernel G_σ, as discribed in the Frangi-Net paper [8].
Given a 2D gaussian kernel G_σ,

$$H_\sigma = G_\sigma * I = \begin{pmatrix} \frac{\partial^2 g_\sigma}{\partial x^2} & \frac{\partial^2 g_\sigma}{\partial x \partial y} \\ \frac{\partial^2 g_\sigma}{\partial x \partial y} & \frac{\partial^2 g_\sigma}{\partial y^2} \end{pmatrix} * I = \begin{pmatrix} H_{xx} & H_{xy} \\ H_{xy} & H_{yy} \end{pmatrix} \tag{5}$$

In our case, the resulting Hessian will be of size $(4 \times 222, 4 \times 222)$, where each of the four hessian components has the same size as the convolved image.
In case of a 2D input image, λ_1 and λ_2 are 2 dimensional tensors that have the same shape as the convolved image as well. Those tensors can be used to compute the Vesselness mask $V_0(\sigma)$:

$$V_0(\sigma) = \begin{cases} 0 & if \quad \lambda_2 < 0 \\ \exp\left(-\frac{-R^2)}{2\beta^2}\right) \times \left(1 - \exp\left(-\frac{S^2}{2c^2}\right)\right) & otherwise \end{cases} \tag{6}$$

Where $S = \sqrt{\lambda_1^2 + \lambda_2^2}$, $R_B = \frac{||\lambda_1||}{||\lambda_2||}$.

The vesselness response is defined in such a way that it is maximal when $\lambda_1 \approx 0$ and $||\lambda_2|| >> ||\lambda_1||$.

Our method was to incorporate prior knowledge to the network by the means of a Frangi filter. Skimage provides a library that directly computes the Frangi mask for a given image [4]. This library was used on our dataset, at the dataloader level, to generate Frangi filtered patches.

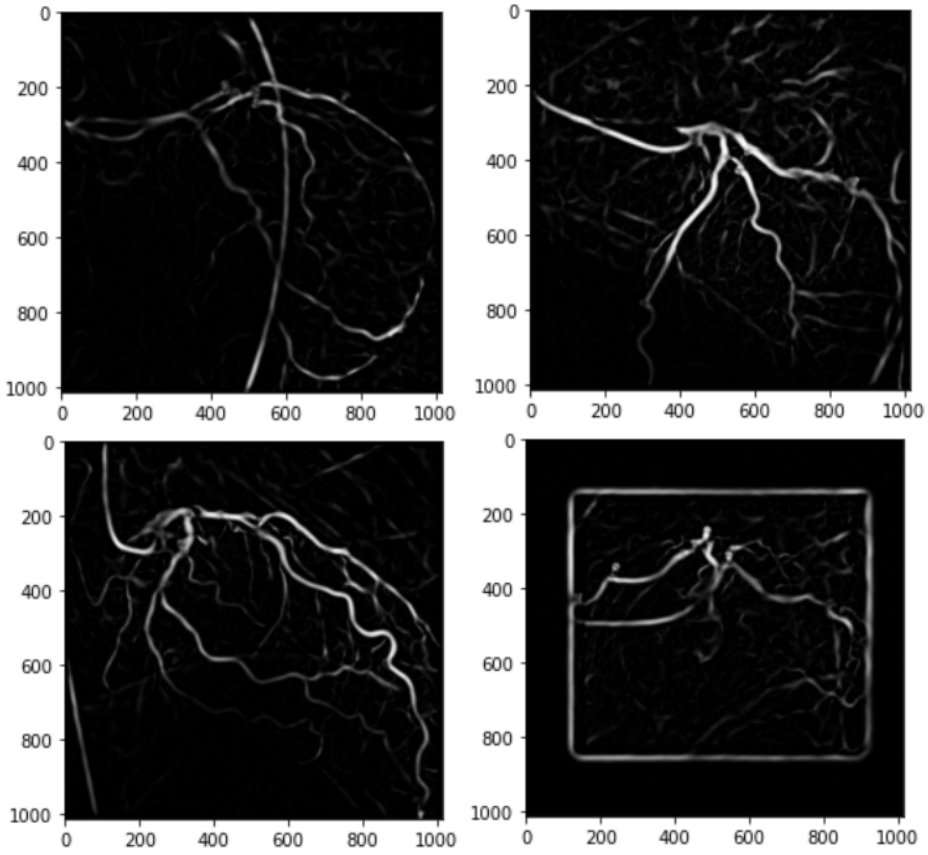

Figure 15: Four different views of Frangi-filtered images from the same patient

The training method that was adopted is to feed the previously-selected model with the Frangi-filter patch as a fourth channel and train the network with the same procedure (Fig.16).

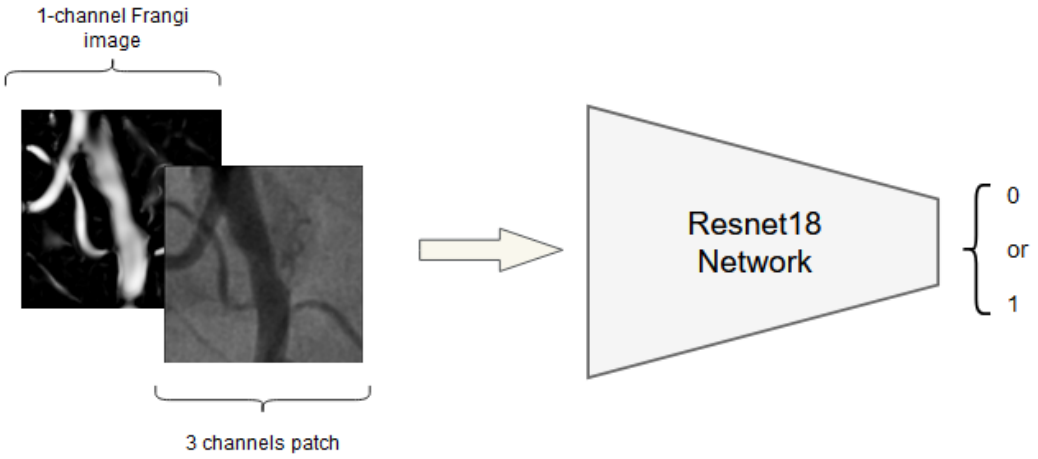

Figure 16: Simple illustration of the training procedure with Frangi images.

For this method, there are extra parameters that need to be tuned for the training such as the filter's scale σ, and the correction constants (α and β) that determine how much the filter is sensitive to deviations from the target structures. Figure 17 shows 3 different filtered patches obtained by applying a Frangi filter to the raw patch with different σ scale ranges $((5,10,5),(10,15,5)and(15,12,5))$, respectively from the left to the right filtered images.

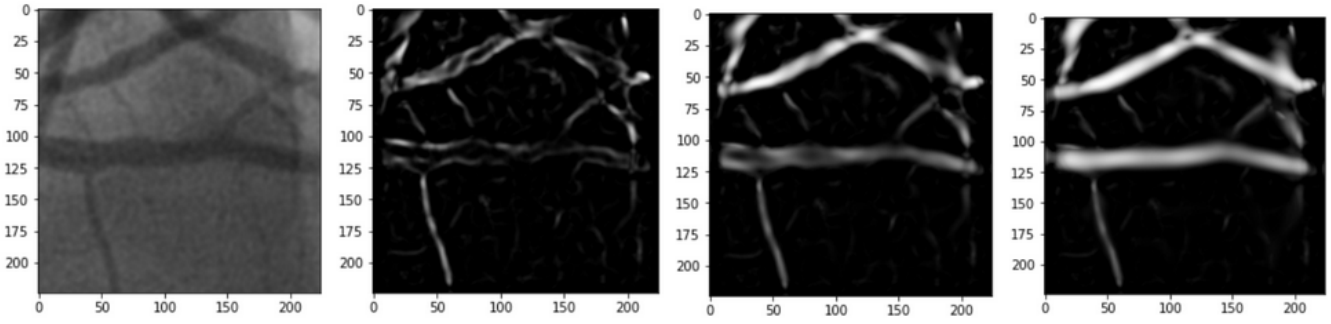

Figure 17: Applying frangi filter with different parameters

For this experiment, the first results showed an overfitting in terms of training accuracy. We therefore decided to run the same 2D cross validation as the one that was used before, in order to investigate the effect of the regularization on the training. From the 3D plots (Fig. 18), the

weight decay does not seem to have high influence, compared to the learning rates, on the training. One can observe that there is a higher f1 variance with respect to the learning rate axis.

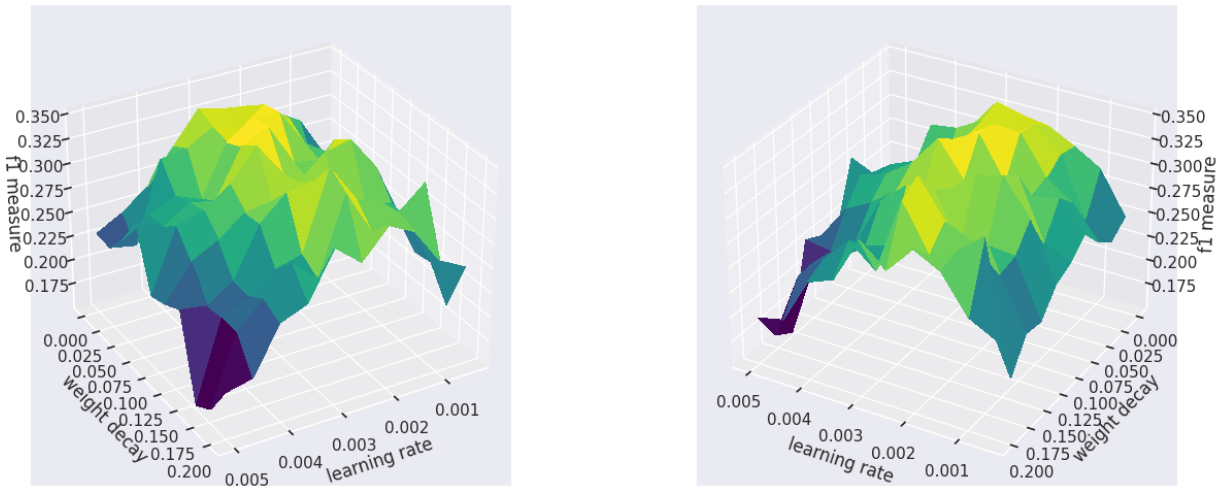

Figure 18: Mean f1-measure with respect to the learning rate and the weight decay for the Frangi model.

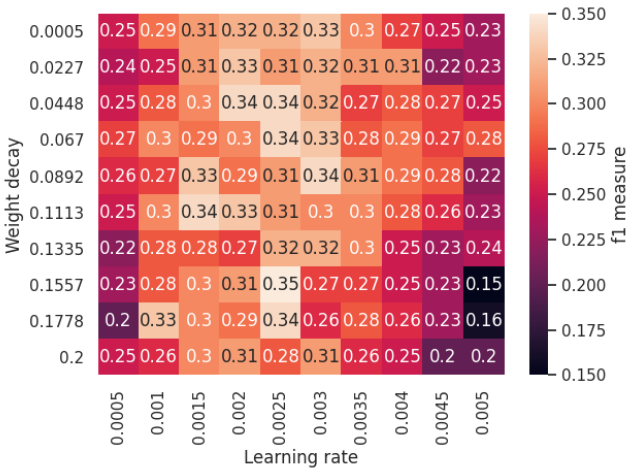

Figure 19: f1-measure heatmap over the learning rate and weight decay for the Frangi model.

Based on the 2D-cross validation results, the final set of relevant hyperparameters that is chosen is the following:

- learning rate = 0.0025
- weight decay = 0.1557

A Results

The best validation accuracy that was reached is 71%, which is slightly below the model that is pretrained on Image-Net. A final comparison will be given in the last section of this report.

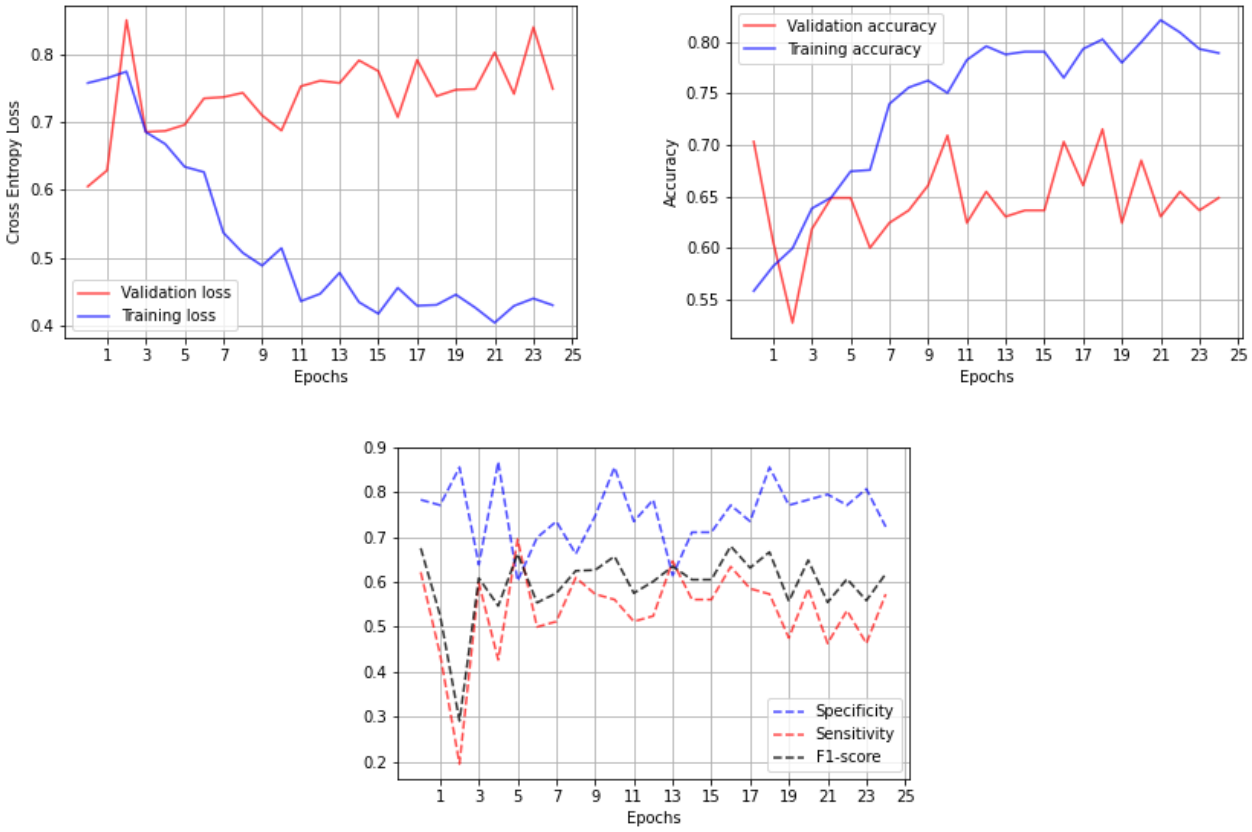

Figure 20: Final results of the Frangi model

6.2 Siamese Nets

"A Siamese neural network is an artificial neural network that uses the same weights while working in tandem on two different input vectors to compute comparable output vectors." [9] Siamese nets usually use the InfoNCE loss, which is a contrastive loss function that is used for self-supervised learning. However, this loss has some drawbacks such as the fact that it requires a large number of negative examples in order to avoid learning trivial solutions. In fact, Siamese nets use weight sharing in the training process. If there aren't enough negative examples, all

input images may collapse to the same latent representation.
Many architectures are used in order to implement a network with such functionality such as SimCLR, BYOL, SwAV and SimSiam. Depending on the implementation, different methods are used to tackle trivial solution problem. For instance, SimCLR requires large batch sizes to be able to have enough negative examples within the batch. Other implementations like SwAV [10] use other techniques such as creating clusters and computing a contrastive loss between the data and the corresponding centroid.
The Siamese nets that were considered in this project are SimCLR and SimSiam for their simple implementation as well as other advantages that will be discussed later in the report.

6.2.1 SimSiam

SimSiam [11] is a Siamese network that maximizes the similarity between two different transformed images that are coming from the same raw image. This implementation is claimed to be more stable than other Siamese nets such as SimCLR (see appendix C), especially when using small batch sizes. SimSiam deals with the collapsing solutions problem by explicitly imposing a stop-gradient operation on one side of the representations as it is shown in the following figure.

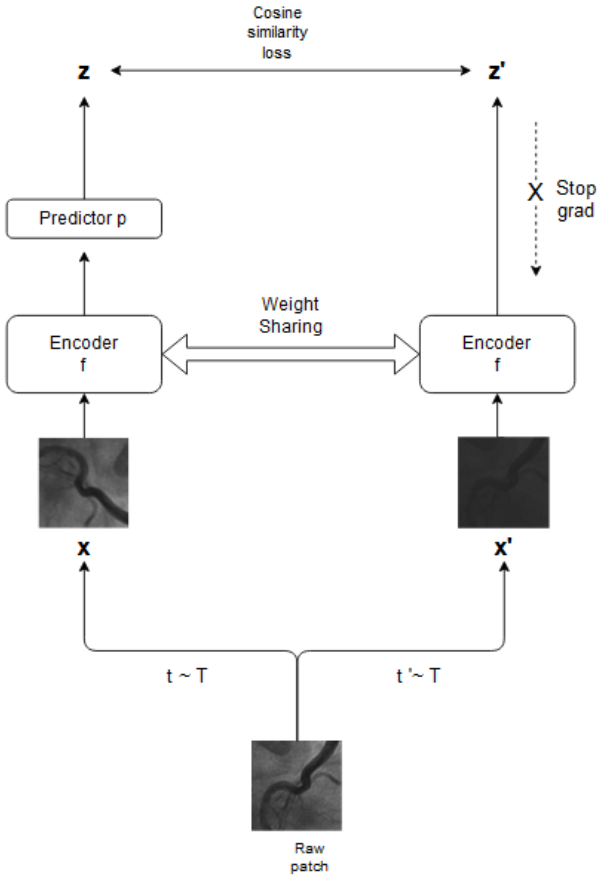

Figure 21: SimSiam network.

Our main motivation behind using SimSiam is to build a robust network that learns important features in the images, independently of the transformations that are being applied.

We used a PyTorch implementation of SiamSiam (`https://github.com/leftthomas/SimSiam`), which was trained and tested on the CIFAR10 dataset. The Encoder f that was used in this implementation is a *Resnet*50 network. The predictor p is a multi-layer predictor that maps $f(x_1)$ to a representation that is similar to $f(x_2)$. Let z be the output of the predictor head p and z' the output of the encoder f for the input x'.

The objective of the pre-training phase is to minimise the cosine similarity loss between both representations z and z':

$$D(z, z') = -\frac{z}{||z||_2} \times \frac{z'}{||z'||_2} \tag{7}$$

6.2.2 SimSiam Results

Figure 23 shows the performance results from the SimSiam training. The most important observation is that the model shows almost no overfitting as the validation loss is able to follow the training loss. Although the model's validation accuracy is smaller than the one we got from the previous results, SimSiam is the best model in terms of sensitivity.

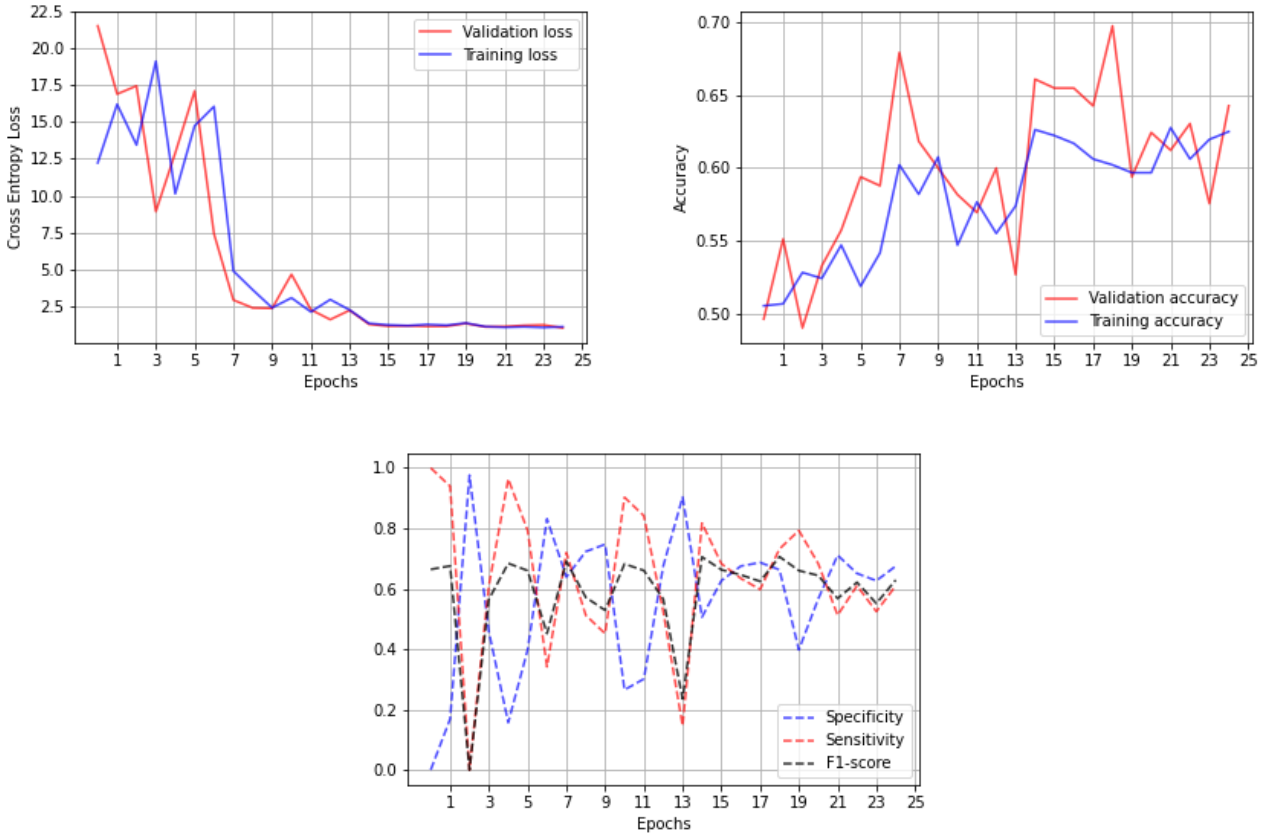

Figure 22: Final results of the SimSiam model

6.2.3 Using Frangi-images as positive examples

Another experiment was tried to learn robust features with the SimSiam network. It was about explicitly maximizing the similarity between the patches and Frangi-positive examples as, the most important features for our task should be located at the vessels level. The training was performed in the same way as for the classical SimSiam, except that the positive example is a 3

channels image composed of the same replicated frangi-filter patch (Fig 23).

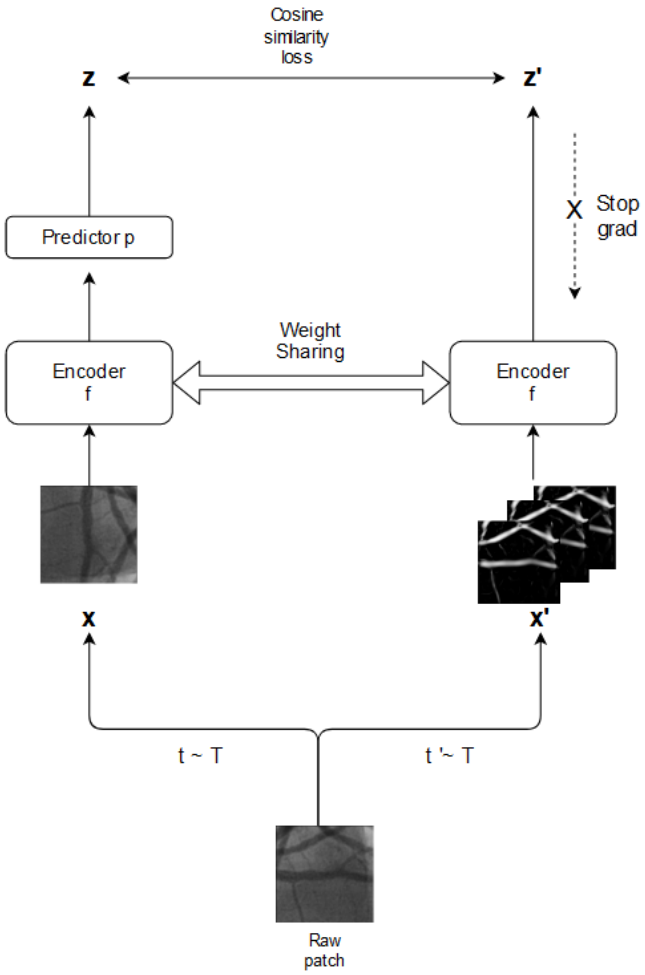

Figure 23: Simsiam Net with Frangi positive samples

The pre-training losses of both Siamese nets reach a plateau after about 50 epochs training (Fig 24). We can observe an offset between the regular SimSiam and the Frangi SimSiam losses. This offset can be explained by the fact that there are more similarities between patches that were transformed using the classical transformations (Fig. 21) compared to the Frangi positives. Note that the minimum possible value of the loss is -1, since we are using a cosine similarity loss.

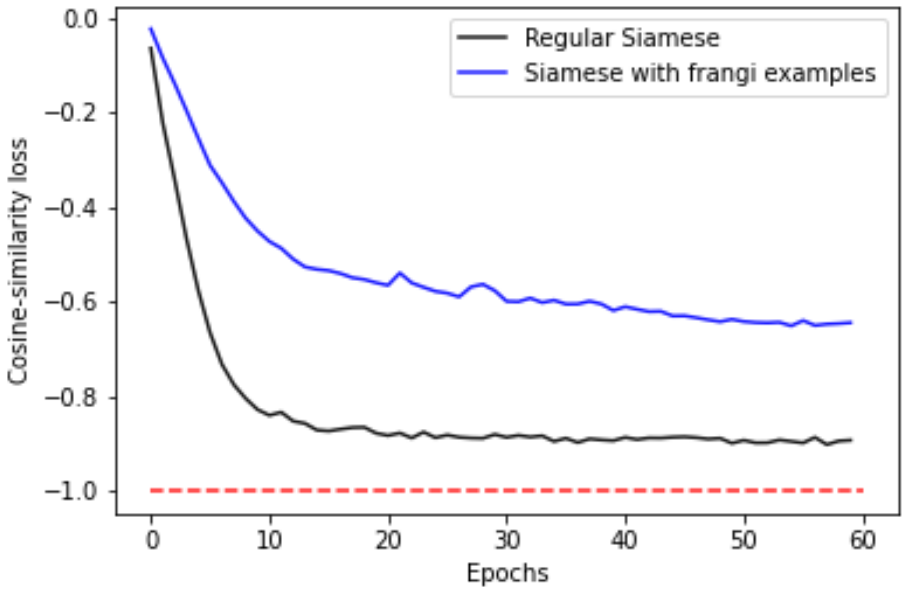

Figure 24: Siamese pretraining losses

6.2.4 Frangi- Simsiam Results

The following figure shows the performance results for the Frangi-SimSiam model. The same behaviour as the previous SimSiam can be observed in the loss curves as well as the sensitivity plot. It is also important to note that the offset from the pre-training phase of both SimSiam models did not lead to significant differences in the final results.

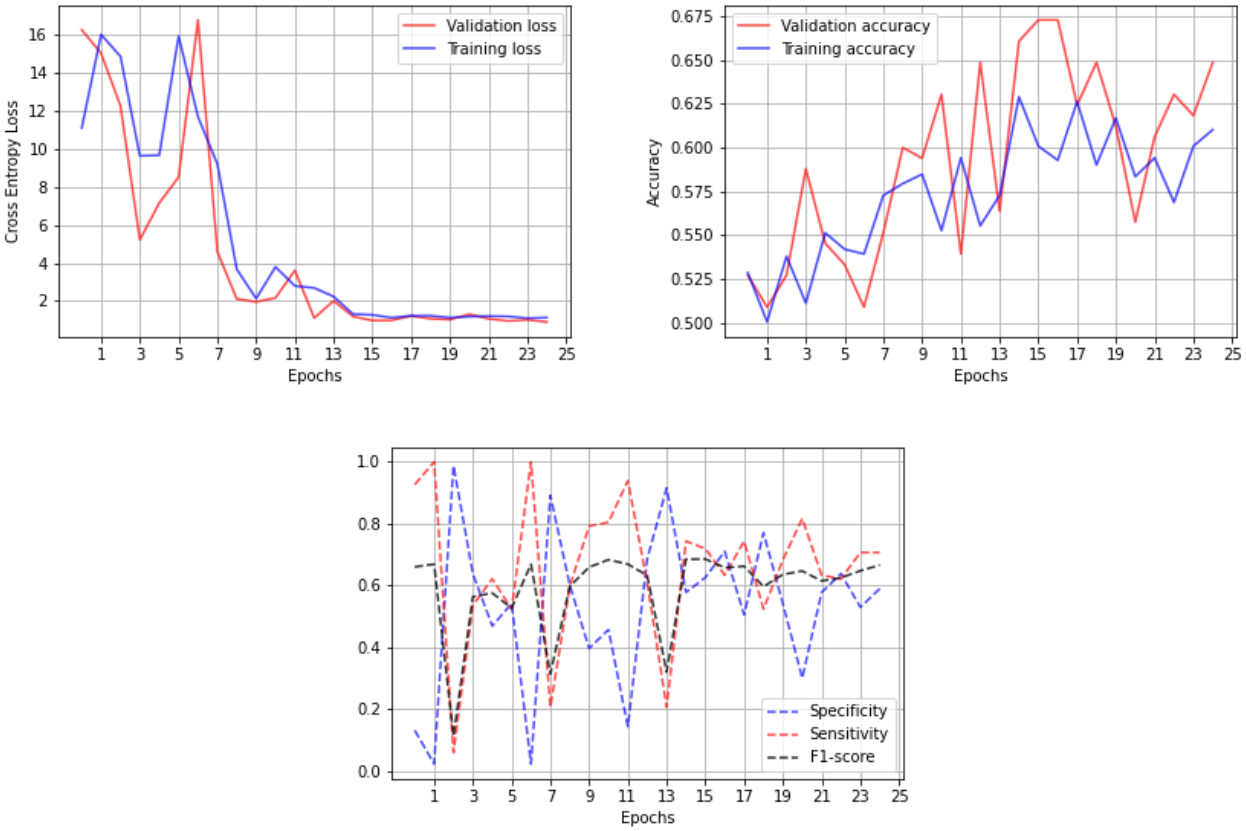

Figure 25: Frangi-Simsiam model final results

7 Models comparison

7.1 Validation and test results

All models were trained and selected based on the best validation accuracy in 25 Epochs training. We don't train for longer epochs as all models' losses saturate around epoch 20. It is important to note that, in this case, the validation accuracy can be considered as a reliable selection criterion since the validation set contains a balanced number of Red and Green patches. Table 2 shows the models' best validation accuracies with the corresponding training times.

Model	Best validation accuracy	Training time
Pre-trained on ImageNet	0.7212	2m 43s
SimSiam	0.6485	6m 41s
Frangi	0.7030	19m 33s
SimSiam with Frangi	0.6545	6m 35s

Table 2: Models performances on the validation set.

The best model, according to this criterion, is the model that is pre-trained on ImageNet, followed by the Frangi model. Siamese models, however, have lower validation accuracies but show a better behaviour in terms of sensitivity.

The Frangi model's training time is relatively large because of the filtered image generation in the dataset class, as well as the inference time that should be longer due to the 4 channels inputs.

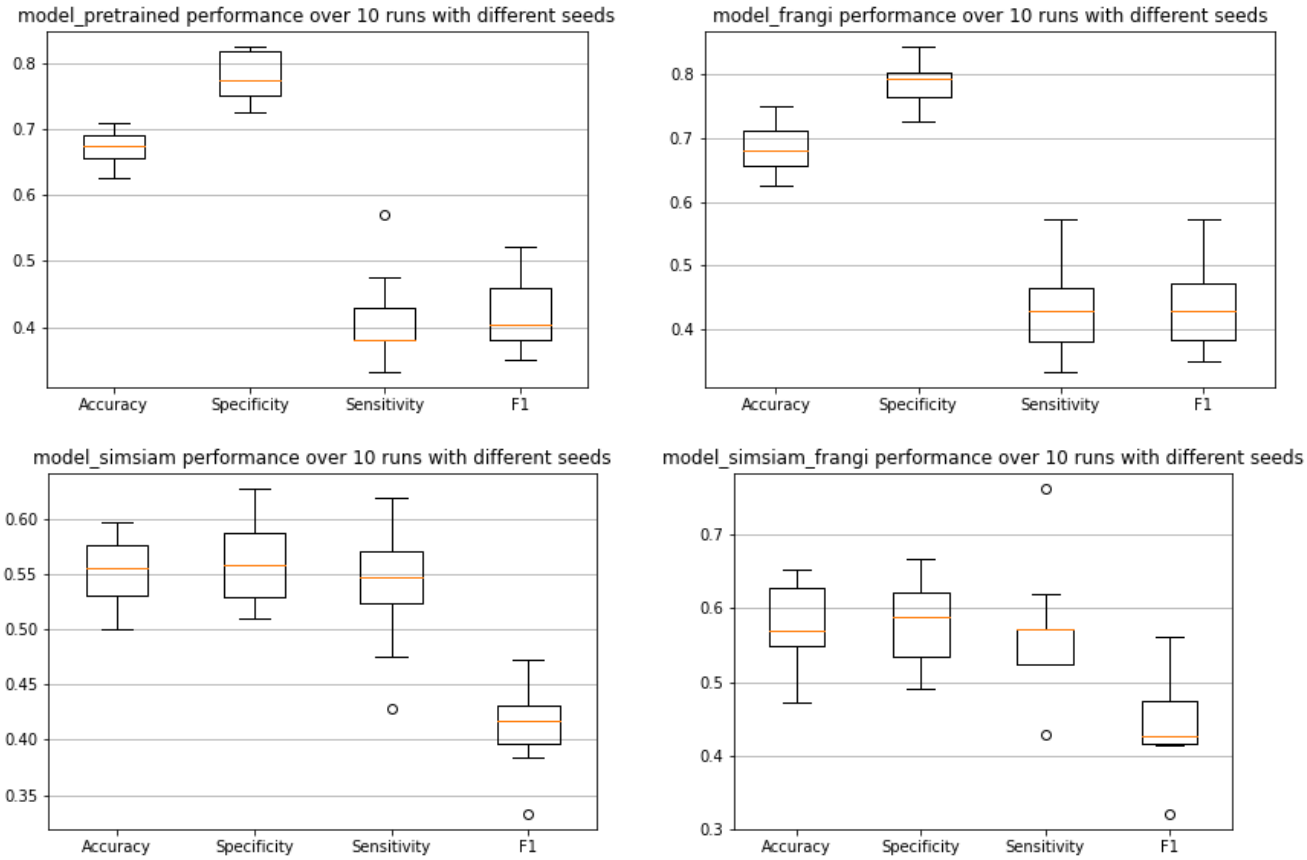

Figure 26: Models test set performance for 10 runs with different seeds

The same models were tested on the test set, keeping the same transformations that were applied in the training phase. Figure 26 shows the evaluation results over 10 runs, with different random seeds. The variance over the runs is coming from the random transformations that are applied to the patches.

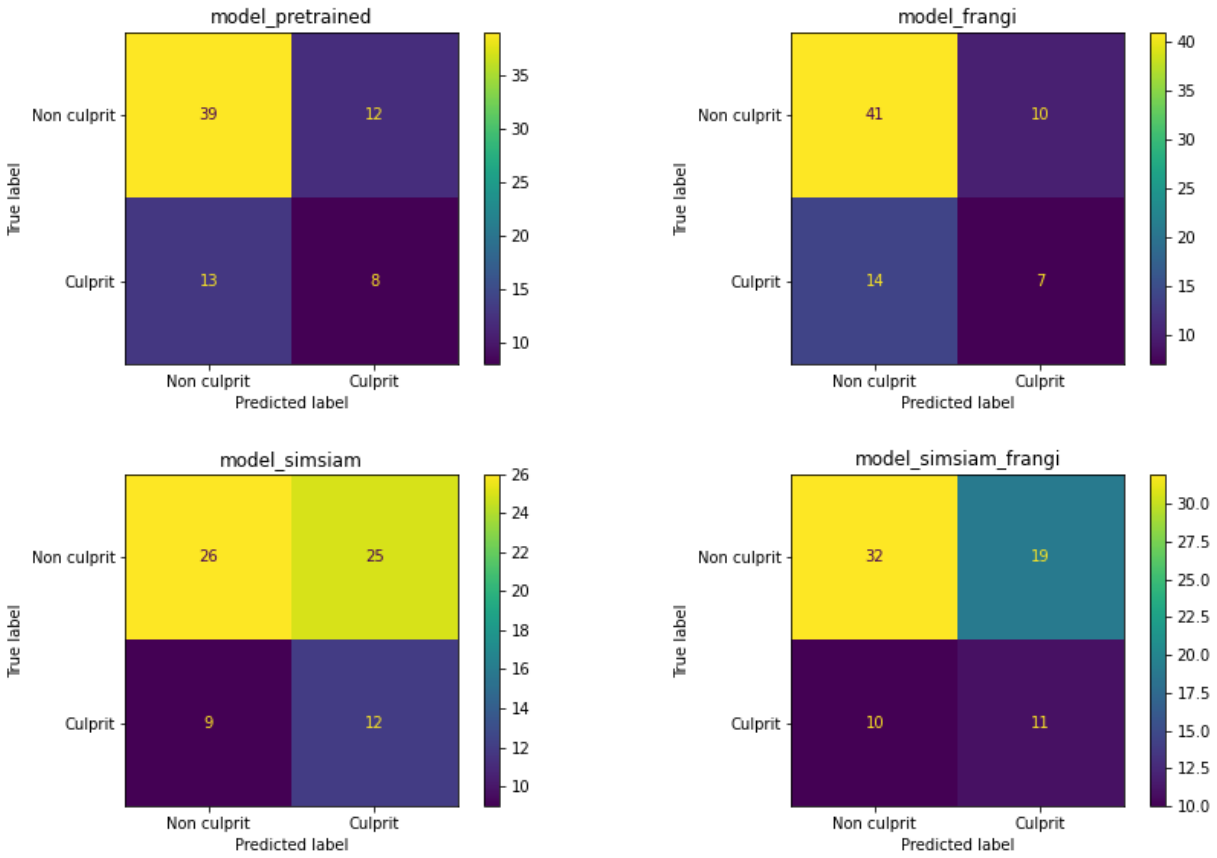

Figure 27: Confusion matrix for each model

In order to test the models with a representative portion of the real data, we did not augment the Red patches in the test set. Therefore, the f1-score can be more reliable than the accuracy in the testing phase. For this reason, we plotted confusion matrices based on the models prediction to help us verify whether the high accuracy is coming from a biased prediction or not.

The model that showed the best average f1-score on the test set is *model_frangi*, followed by the Siamese models and then *model_pretrained*, which has slightly lower f1-score but yields better average testing accuracy. Both Siamese models had poorer accuracy results but a relatively large F1 score due to the high sensitivity, which is coherent with the training and validation results of Siamese nets (Fig. 22 and Fig. 25).

The models *model_pretrained* and *model_frangi* have close performances in the prediction task and seem to be the best according to the results from both the validation and testing phases. Siamese models, on the other hand, showed a good behaviour in terms of sensitivity, which can

be very promising. In fact, detecting positives is often more important in medical prediction tasks. Therefore, Siamese nets could be suitable for our application.

Overall, the classical resnet and the frangi model are the best models to choose, if one cares more about the testing accuracy. Whereas, Siamese networks could be more suitable when the sensitivity is more important.

7.2 Patches inspection

Figures 28 and 29 show, respectively, the set of test patches that were the most difficult to classify as well as the patches that were correctly classified by all the models.

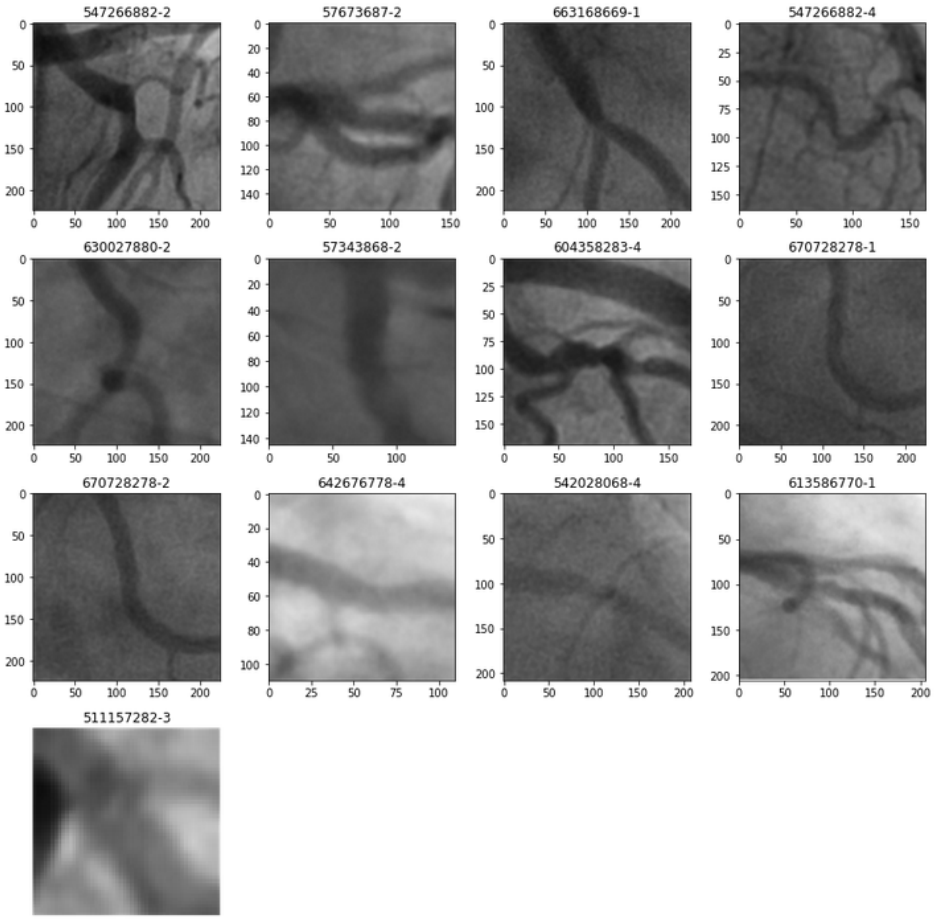

Figure 28: Misclassified patches

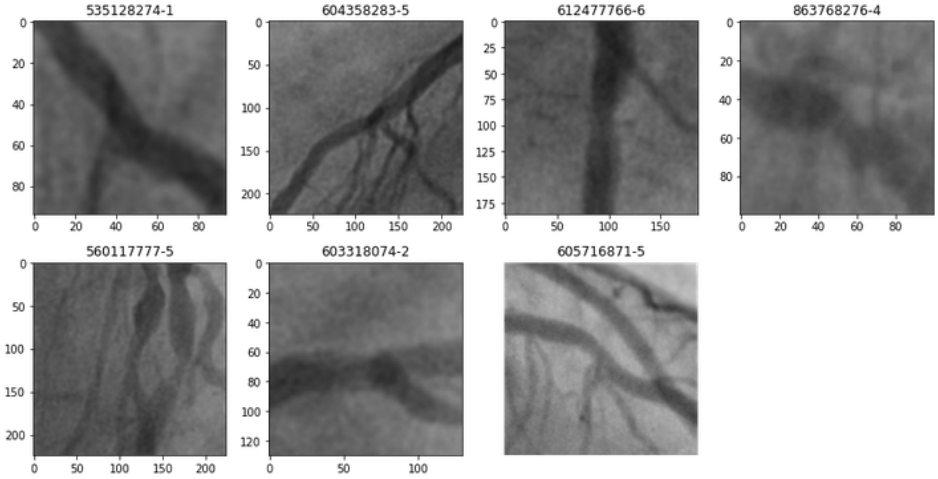

Figure 29: Correctly classified patches

In orther to further inspect the misclassified patches. Heatmap visualizations were displayed on each misclassified patch to highlight the models attentions in the evaluation phase. The visualizations are provided for the model that is pre-trained on ImageNet as well as the Frangi model (see appendix B).

The first observation is that most of the misclassified patches belong to the *culprit* class (9 *culprit* vs 5 *nonculprit*). This is coherent with the previous results since all models have higher specificity than sensitivity. One could also observe that, except for Patch number 1 and 71, both models seem to focus on vessel regions in the patches. The common point between patches 1 and 71 is that they have relatively smaller size (65x65 and 51x51), compared to the other patches. Resizing the patch to 224x224 might add important noise to the image, and thus, make both models focus on irrelevant regions. If we have had a larger dataset, dropping detected patches of a size that is less than a certain threshold (e.g 150x150) could be a solution to this issue.

The results from Figures 28 and 29 were sent to cardiologists in order to see if they can observe any specific pattern on the misclassified and correctly classified patches. The feedback was that there is no significant pattern that can be observed, except that quite often, the culprit patches contain "a part where the vessel is somewhat blurry (with a grey which is a little bit light as compared to before and after)". And that these parts are always suspicious in their clinic. Which means that the models were not able to detect such a pattern.

A potential way to improve the models sensitivity to such patterns is to tune the Frangi Gaussian variance σ in the frangi model, since the Filter is very sensitive to this parameter (as shown in figure 17). It could also be interesting to combine a Frangi-Net (see appendix D) with the regular *Resnet*18 architecture in order to learn the σ parameter in the training phase.
Trying these experiments while keeping the same test set could give biased results. It is therefore important to make a different train/test split or even add a new dataset, when trying the aforementioned experiments.

8 Future work

Future work may include the integration of a Frangi-Net to the classical *Resnet*18 architecture in order to learn Frangi parameters, as specified in the previous section.

One way that could improve the SimSiam performance is to use artificial data in the pre-training phase. It will not only add more samples, and thus, more negative examples to the pre-training, but it can also help model focus on the stenosis part of the data. As explained in appendix A, the artificial data is a set of coronary angiography patches, labelled with *stenosis* or *notstenosis*. These labels, can be used in an intermediate level of the SimSiam network. One could, for instance, add a multi-layer projection head that maps the latent representations within the Siamese net to a 2 dimensional output, while keeping the same initial projection head (as illustrated in figure 30). The two dimensional output can be used to define an auxiliary task (predicting stenosis or not) by computing an auxiliary loss (e.g. BCELoss between the output and the true label), as mentioned in the GoogleNet paper [12]. The total loss could be defined as a weighted sum between the cosine similarity loss and the auxiliary loss. This may help the network learn more relevant features.

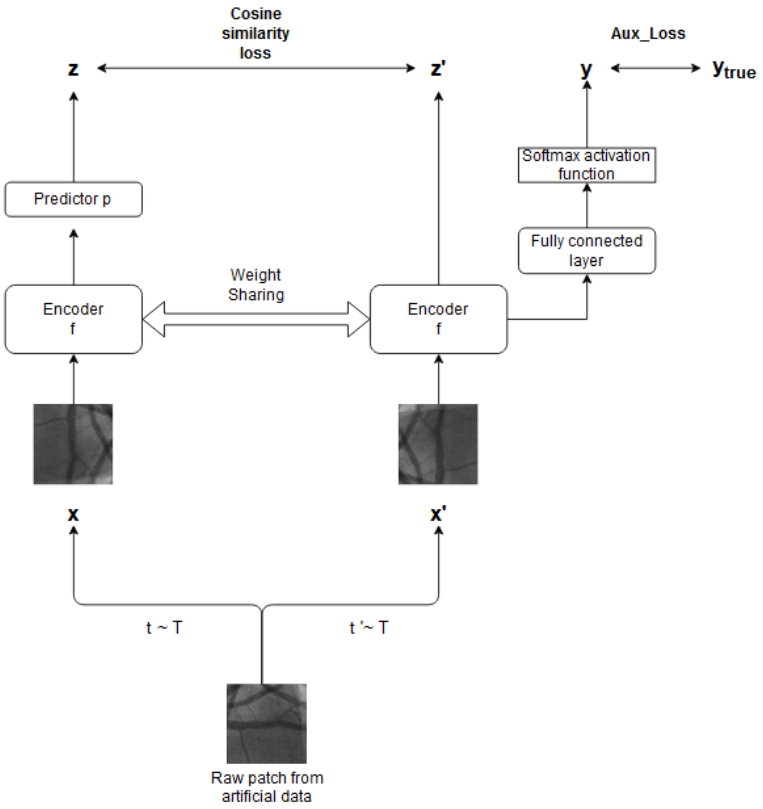

Figure 30: Illustration of Siamese implementation with auxiliary loss.

Finally, another experiment that could be interesting to try, is to compute the predictions based on a majority vote from all the models. In fact, the previous results showed that the models behaviours are quite different, which may help yielding more accurate predictions.

9 Conclusion

In this project, we have considered a new prediction task on angiography images. Our goal was to build a robust model capable of yielding meaningful predictions that would eventually assist cardiologists in their diagnosis. This task was not trivial as many challenges needed to be addressed, such as the limited and imbalanced data set. Therefore, many experiments were explored, from the data preparation to the model's robustness improvement. The first conclusion, that we can draw from this project, is that the data balancing is crucial when dealing with limited dataset. The results showed a significant improvement in the models' performances after balancing the data, which allowed us to investigate even more advanced aspects in the project.
Many methods were tested to help the baseline model focus on more important features such as using the contrastive learning in Siamese networks or by including Frangi-filtered data.
While the classical Resnet18 and Frangi model showed a better performance in terms of validation and testing accuracy, SimSiam Nets yielded poorer accuracy, but were better at detecting culprits.
The Frangi filter is a commonly used tool to assist models in the training, when dealing with vessels-like images. Our Frangi method did not show significant improvements in the training. Although this could be very promising if tuned properly. Considering the doctors feedback about the grayscale level in the heart vessels, this method could be very important to help the model detect such features.
Finally, there is still a future work that could be done, especially if a new dataset is available, such as combining the regular model with a Frangi-Net or training a Siamese Net, by including prior knowledge through an auxiliary loss.

A Model pre-trained on artificial data

Another interesting experiment was done during the final stages of the project. It was about training a model that is pre-trained on artificial cardio data. The pre-training was a classification task that, given an input patch, predicts if the patch contains a stenosis or not. Figure 32 shows promising test results of this model. This pre-training approach can be combined with a Siamese net by including another multi-layer projection head in the SimSiam architecture and implementing an auxiliary loss that can be used in the $stenosis - or - not$ classification, before the final prediction.

Running a cross validation on this model might also be a good experiment, as the presented results (Fig. 31) were computed without any hyperparameters tuning.

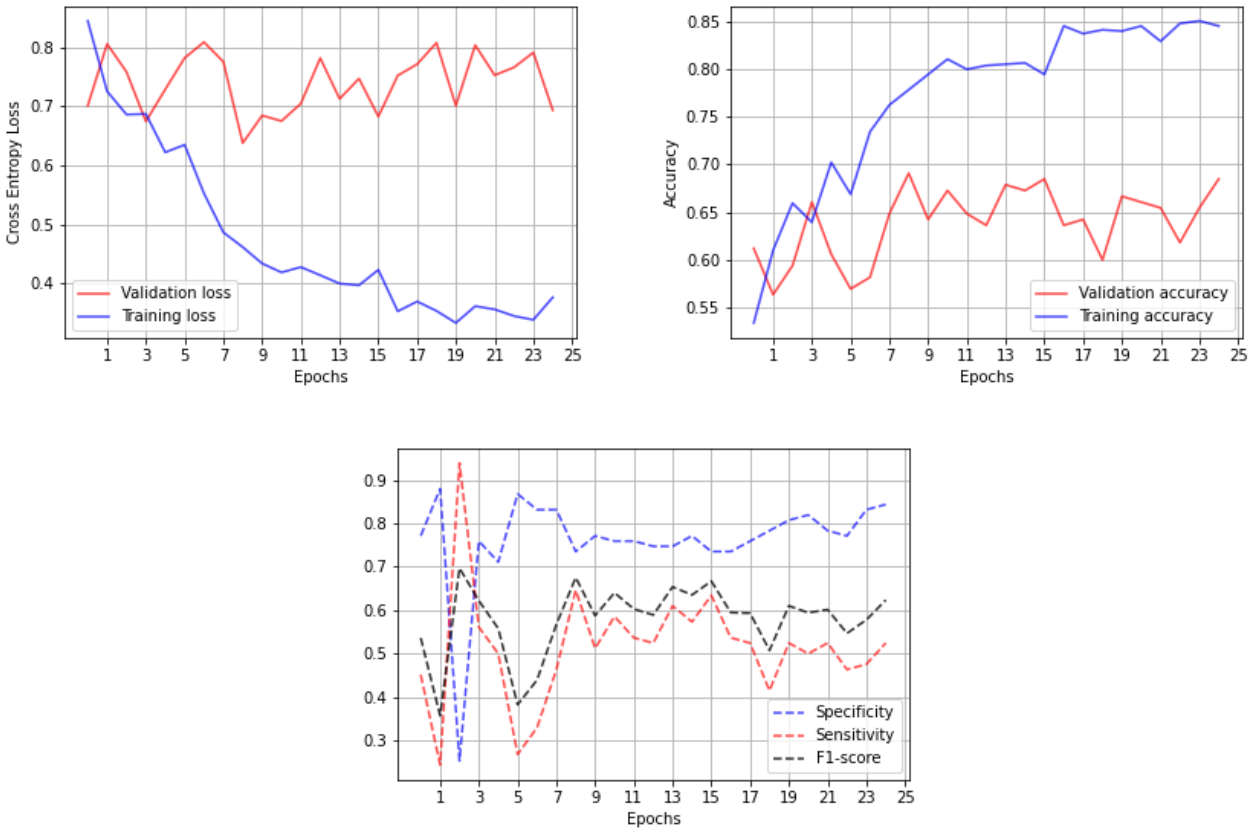

Figure 31: Final results of the model that is pre-trained on artificial cardio data

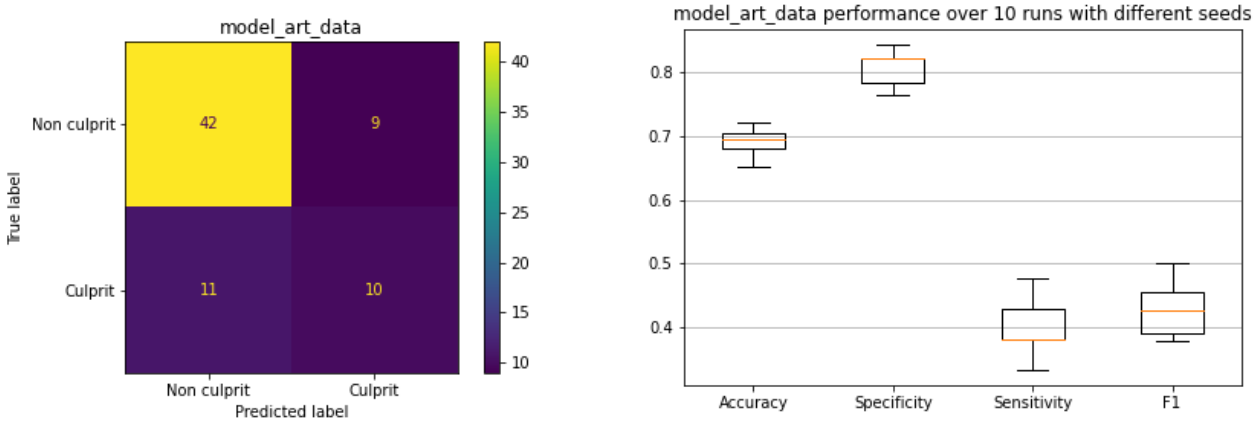

Figure 32: Test results of the model that is pre-trained on artificial cardio data

B Heatmap visualizations

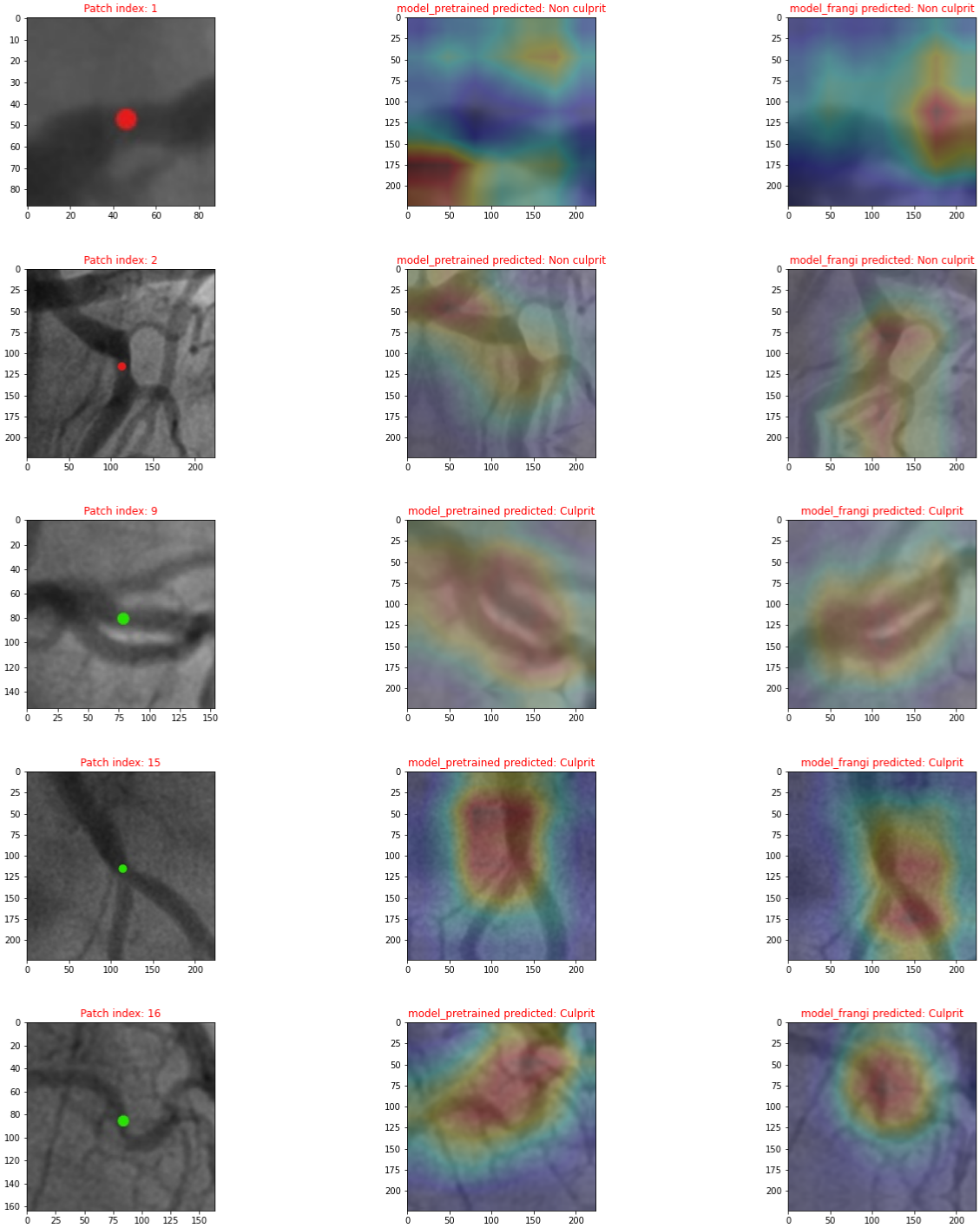

Figure 33: Heatmap visualization of misclassified patches - Part I

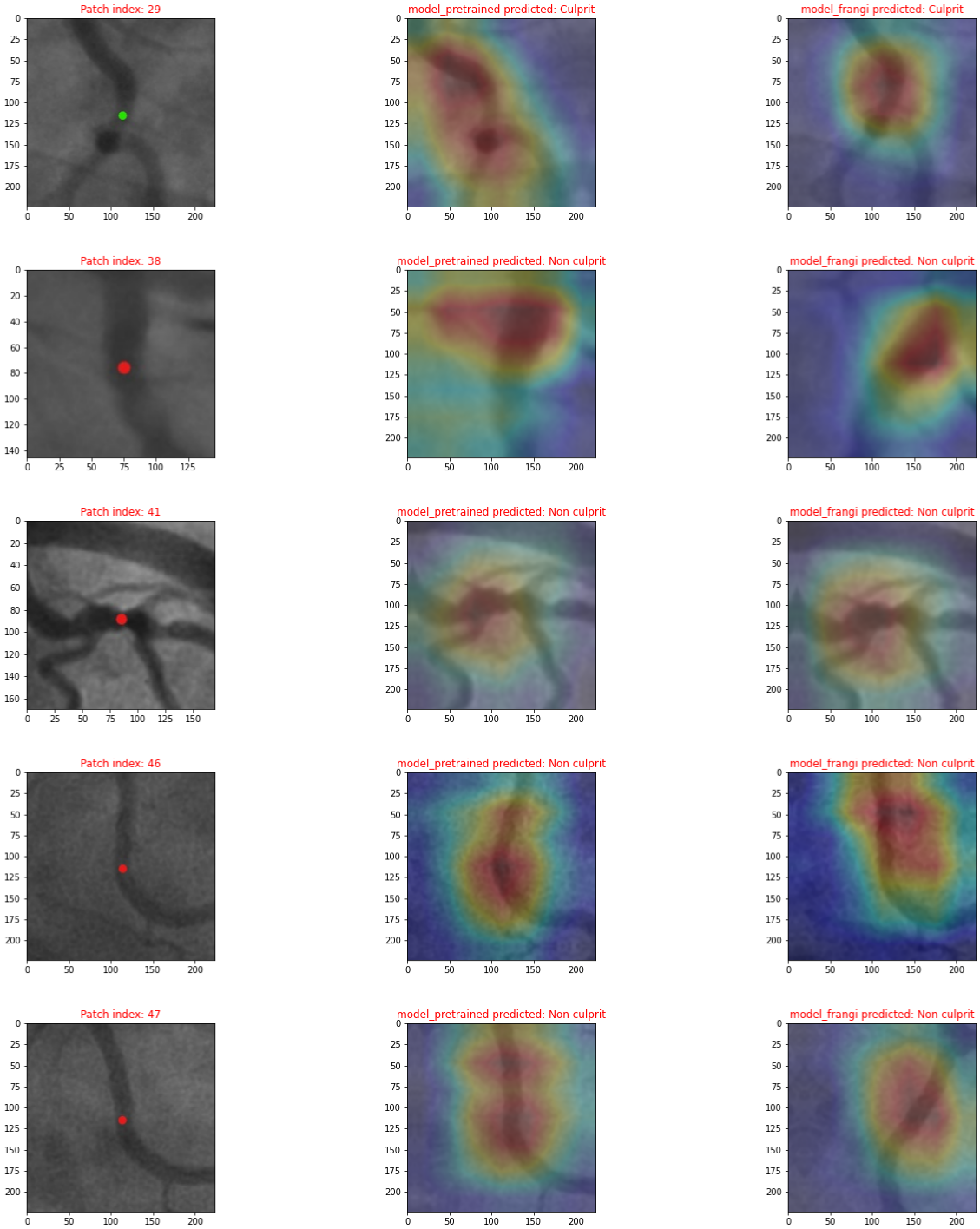

Figure 34: Heatmap visualization of misclassified patches - Part II

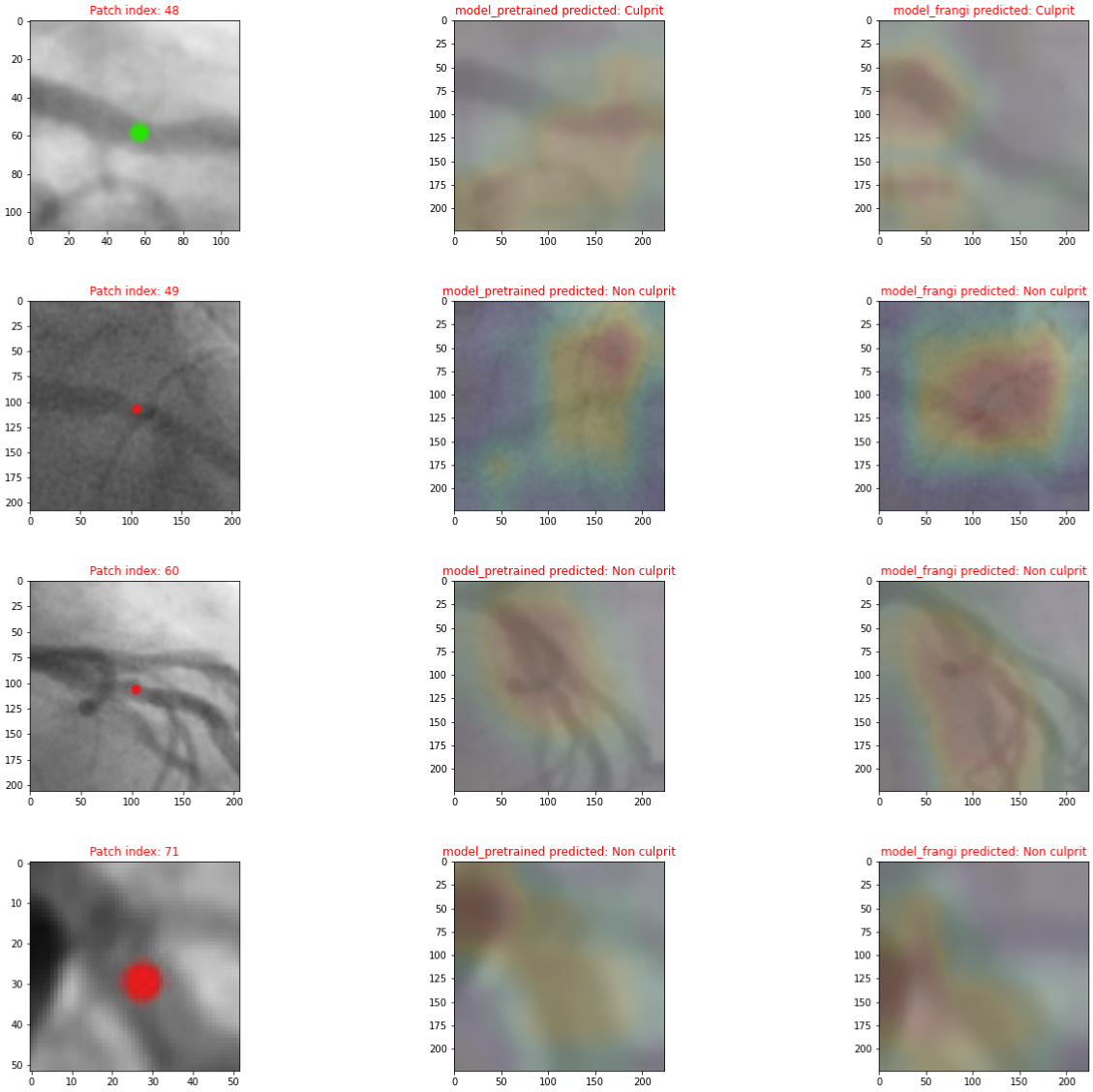

Figure 35: Heatmap visualization of misclassified patches - Part III

C SimCLR

C.1 Unsupervised SimCLR

The first experiments with Siamese nets were performed with the SimCLR architecture [13]. To do so, we adapted a pyTorch implementation of SimCLR (see `https://github.com/sthalles/SimCLR`) to our dataset.

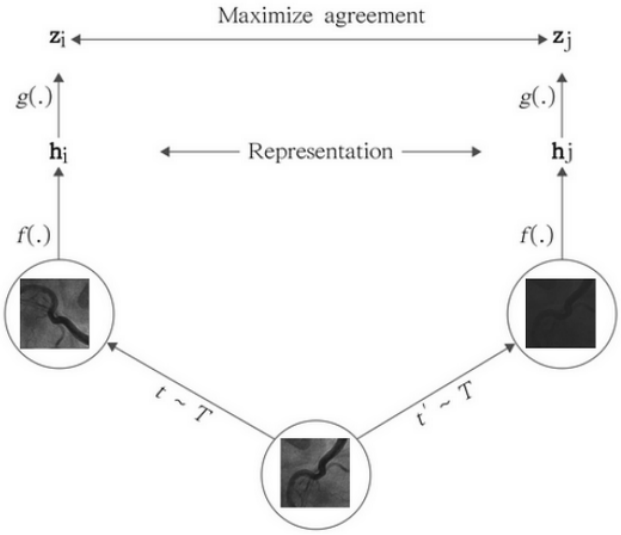

Figure 36: SimCLR pre-training phase. Figure inspired from the SimCLR paper.

The implementation uses a *Resnet*18 network as a backbone and a multi layer projection head g, composed of a fully connected layer with a ReLU activation function. For a given batch of N examples, a contrastive loss can be defined for each pair (i, j) as:

$$l_{i,j} = -log(\frac{\exp(sim(z_i, z_j)/\tau)}{\Sigma_{k=1}^{2N} \mathbb{1}[k \neq i] \exp(sim(z_i, z_k)/\tau)}) \tag{8}$$

Where $sim(.,.)$ is the cosine similarity function. Note that all the other$2(N-1)$ transformed samples are considered as negative examples.

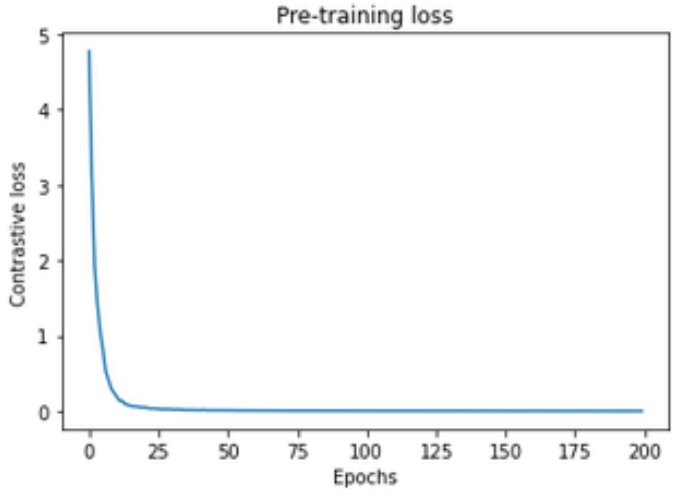

Figure 37: Pre-training loss of the SimCLR model trained on the Cardio Dataset.

The results were not very promising, as the model was always predicting $non - culprit$ in the final training phase, even if the pre-training loss converged to a very small value (Fig. 37). This could be due to the small number of the negative examples that are contained in the batch since we couldn't increase the batch size as our dataset is limited. This small number may cause an instability in the sense that the network may learn trivial solutions.

C.2 Supervised SimCLR

In order to check if the learned features are meaningful, we tried a supervised version of SimCLR [14]. This version uses the labels in order to maximize the similarity between instances of the same class and maximize the dissimilarity between instances of different classes. In other words, the Supervised SimCLR puts the representations of images coming from the same class closer together in the embeddings space.

We performed a TSNE on the learned features of the embeddings space in order to see if both classes are separated. The following image confirms that the learned features are not relevant for our task, as both classes are overlapping.

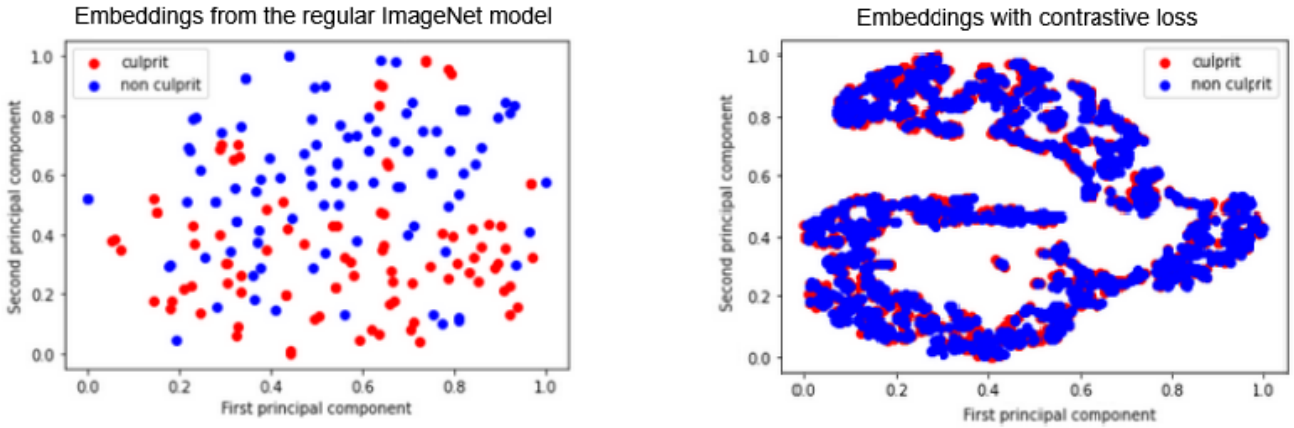

Figure 38: Embeddings comparison between different models

A potential explanation of this could be that the transformations that we can use for the cardio dataset, without losing meaningful features, are very limited and may not be very significant for the contrastive learning. Which may make the model focus more on common features between both classes such as "recognising vessels" rather than "measuring the narrowing of the vessel".

- Transformations applied to a patch from the cardio dataset
- Transformations applied to an image from CIFAR10 dataset

Figure 39: Comparison between the transformation that were applied to the cardio and CIFAR datasets.

Another potential explanation could be that the cardio interclass similarity is relatively high compared to the *CIFAR*10 dataset (Fig. 40), which makes the dissimilarity less significant in our case.

Figure 40: Comparison between the cardio dataset and the CIFAR dataset.

D Frangi-Net

An attempt to implement a custom Frangi-Net with he PyTorch framework was done to learn the Gaussian variance σ as a training parameter in the training phase. In order to verify if the custom Frangi-Net has been correctly implemented, a sanity check was performed, by considering the filtered "camera man" example that is given in the *skimage* Frangi documentation as a target result and compare it to our network's output. The figure bellow shows a comparison between the outputs of our custom Frangi-Net and the one that is computed by the $frangi()$ function, from the *skimage* library [4].

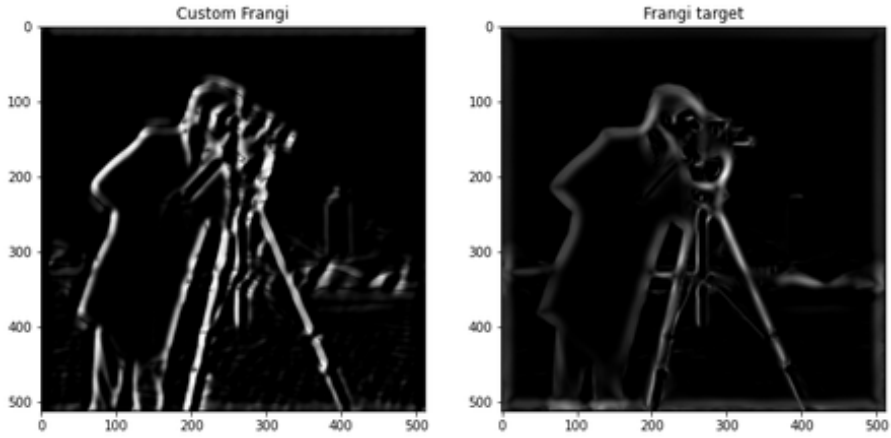

Figure 41: Comparison between the custom Frangi-Net and the $frangi()$ function outputs.

Figure 42 shows the output image after applying each operation, as described in section 6.1.

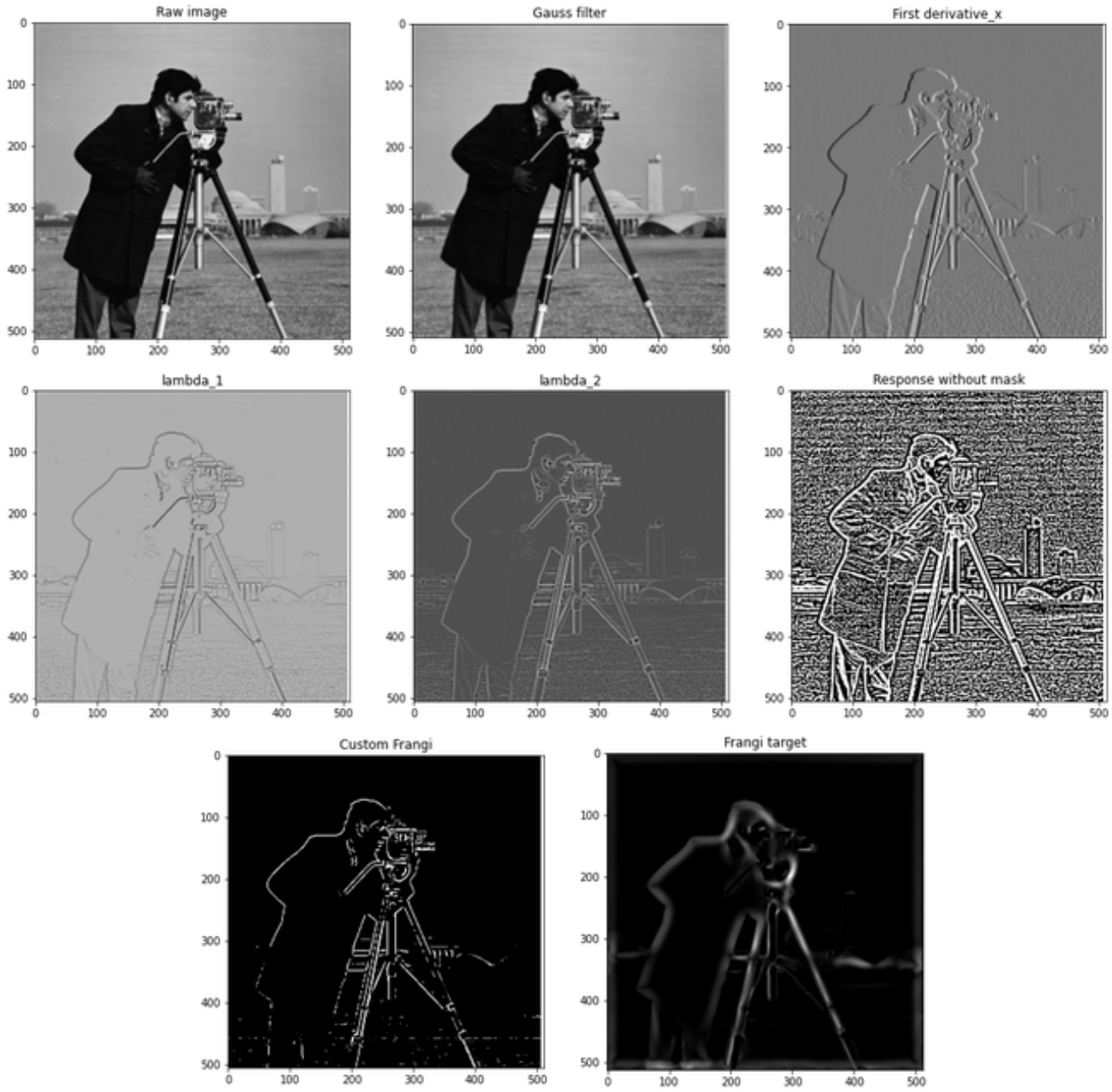

Figure 42: Sequence of results after each operation that is applied in the Frangi-Net.

E Visualization functions

A visualization function was implemented to be able to asses the models performances, interactively, by specifying the patient ID and the model to be evaluated as well as other parameters. This function displays all views with all different patches for a given patient as well as the corresponding label and prediction.

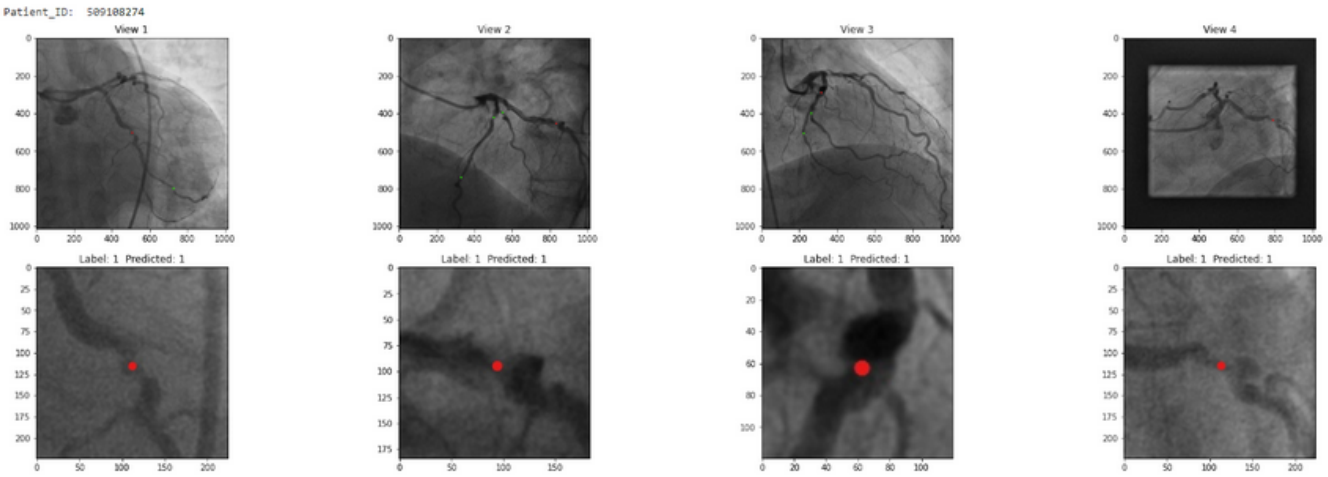

Figure 43: Results for patient Patient_ID: 509108274

F Codes

The implementations can be found in `https://github.com/Omarraita/MI_prediction`. The instructions to run the codes are included in the README file of the same repository.

References

[1] "Myocardial infarction," June 2021. https://en.wikipedia.org/w/index.php?title=Myocardial_infarction&oldid=1026458252.

[2] L. Xie, H. Zhang, X. Liu, X. Wang, D. Chen, Y. Xu, Z. Sun, W. Zhou, L. Song, C. Guan, *et al.*, "Automatic and multimodal analysis for coronary angiography: training and validation of a deep learning architecture.," *Eurointervention: Journal of Europcr in Collaboration with the Working Group on Interventional Cardiology of the European Society of Cardiology*, 2020.

[3] J. H. Moon, W. C. Cha, M. J. Chung, K.-S. Lee, B. H. Cho, J. H. Choi, *et al.*, "Automatic stenosis recognition from coronary angiography using convolutional neural networks," *Computer methods and programs in biomedicine*, vol. 198, p. 105819, 2021.

[4] "Frangi filter — skimage v0.14.3 docs."

[5] S. Yang, J. Kweon, J.-H. Roh, J.-H. Lee, H. Kang, L.-J. Park, D. J. Kim, H. Yang, J. Hur, D.-Y. Kang, *et al.*, "Deep learning segmentation of major vessels in x-ray coronary angiography," *Scientific reports*, vol. 9, no. 1, pp. 1–11, 2019.

[6] "Albumentations Documentation." https://albumentations.ai/docs/.

[7] H. A. Nugroho, R. A. Aras, T. Lestari, and I. Ardiyanto, "Retinal vessel segmentation based on frangi filter and morphological reconstruction," in *2017 International Conference on Control, Electronics, Renewable Energy and Communications (ICCREC)*, pp. 181–184, 2017.

[8] W. Fu, K. Breininger, T. Würfl, N. Ravikumar, R. Schaffert, and A. Maier, "Frangi-net: a neural network approach to vessel segmentation," *arXiv preprint arXiv:1711.03345*, 2017.

[9] "Siamese neural network," Apr. 2021. Page Version ID: 1020522415.

[10] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," *arXiv preprint arXiv:2006.09882*, 2020.

[11] X. Chen and K. He, "Exploring simple siamese representation learning," *arXiv preprint arXiv:2011.10566*, 2020.

[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[13] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, PMLR, 2020.

[14] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *arXiv preprint arXiv:2004.11362*, 2020.