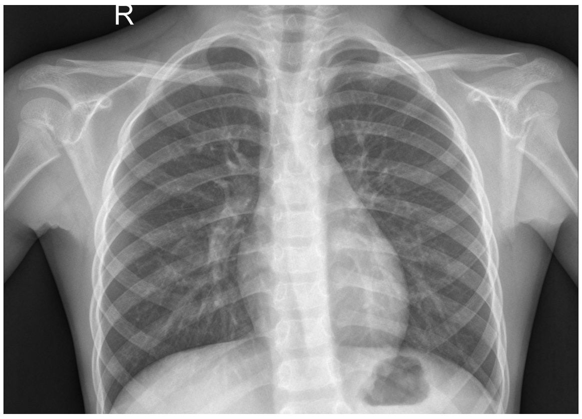
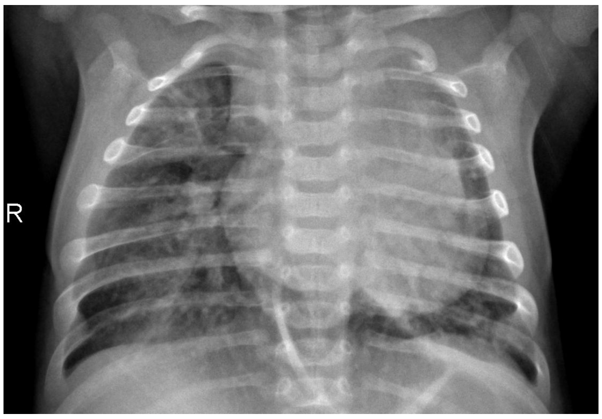
Pneumonia Detection from chest X-Ray images

1) Problem presentation

Pneumonia is an infection of the airways caused by a bacterium or a virus. It is responsible for more than 16,000 deaths in France every year. And it’s a burning issue : when COVID-19 attacks the lungs, it may cause fatal pneumonia. If it is not quickly detected and treated, it can lead to Acute Respiratory Distress Syndrome (ARDS). This condition is responsible for 30% mortality in patients on artificial respirators. The rapid and effective detection of pneumonia is therefore an essential issue in the survival of the patient, but also in the management of hospital beds equipped with artificial respirators in these times of health crisis. A pneumonia quickly diagnosed is an extra bed for the people who need it. We will base our detection on X-Ray images of chests and use MLP and SVM classifiers. You can see below two images of chests : one of a healthy patient, the other of a pneumonia suffering patient.



Healthy patient



Pneumonia suffering patient

As we can see, the difference between a healthy and sick person is very slight but still exists. On the pneumonia suffering patient image, we can see areas of abnormal opacification in the image : we are trying to detect these areas.

2) Tools and methods used

As they are lots of ways to implement an estimator, we decided to divide the work in two axes :

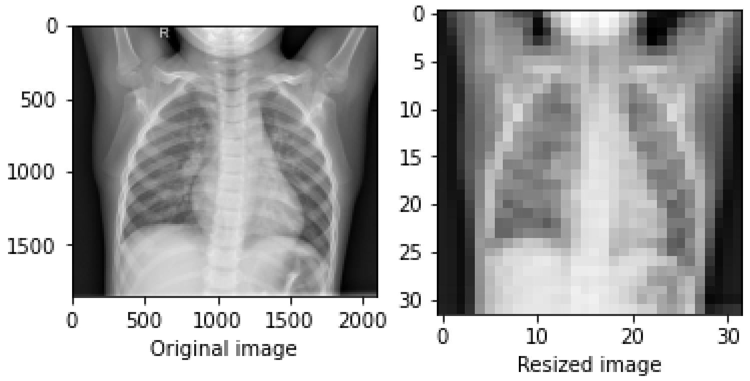
- Alexis worked with methods that we implemented ourselves (ie forward, back-propagation, accuracy and loss methods inspired from lab2 neural network methods) and worked on the dataset to improve the test accuracy (resize, cropping, convolution, brightness).
- Maelle worked on a MLP(multilayer perceptron) classifier and a SVM(super vector machine) model of the machine learning libraries scikit-learn. Then, to decide what is the best model, she did model selection.

This section presents what we did and implemented and explains how we progressed during our project. You can download the database we used by clicking here, and the code here.

3) Preparing the dataset

3.1) Loading and resizing the images

We have more than 2800 X-RAY images of chests, classified as NORMAL or PNEUMONIA. We use the os library to load the images. As there are approximately 1300x1000 pixels we decided to resize them in 32x32 to diminish the time of execution of our algorithm. Some images were not in black and white : we had to convert them. Then, we transform the picture in an array. Each element of the array represents a pixel by a number between 0 (black) and 255 (white). Finally, we normalize our input by dividing every coefficient by 255; and we reshape our matrix as a single line of coefficient belonging to [0,1].

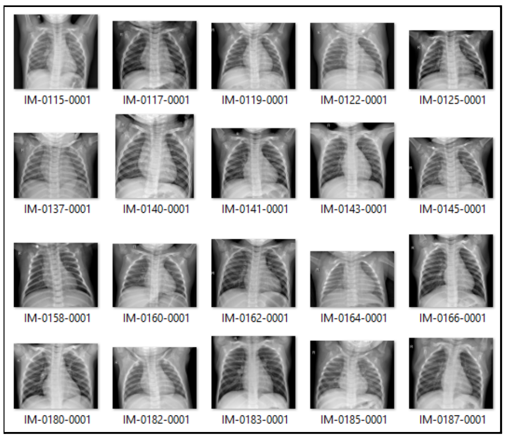


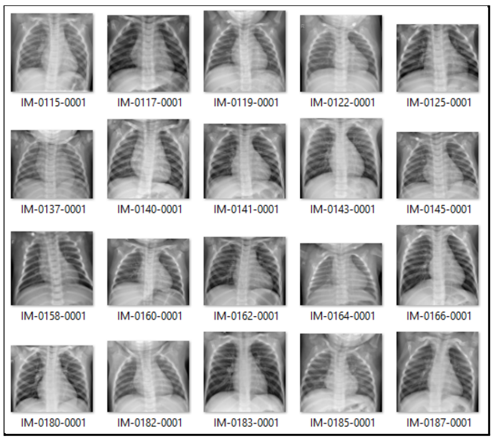
3.2) Cropping Algorithm

As we can see above, the "resize" operation implies some distortion of the picture as we force it to become a square. Moreover, if we take a look at the data, all the images are not framed in the same way. Some of them have a black margin on the right, on the left or on the upper side, between the face and the arms. But some of them don't. These informations are not useful and might disturb our algorithm so we managed to create an algorithm to crop and uniformize these images.

The algorithm is quite simple : it runs every line/column starting from a side. If there is a certain number of consecutive pixels that are black (i.e pixel <70), the algorithm adds 1 to the count of lines/columns that should be erased and goes to the next line to check. The algorithm stops when the condition above is not found and crops the image. We applied this algorithm to all the images of the train and test folder to create the train1 and test1 folder containing all the cropped images.

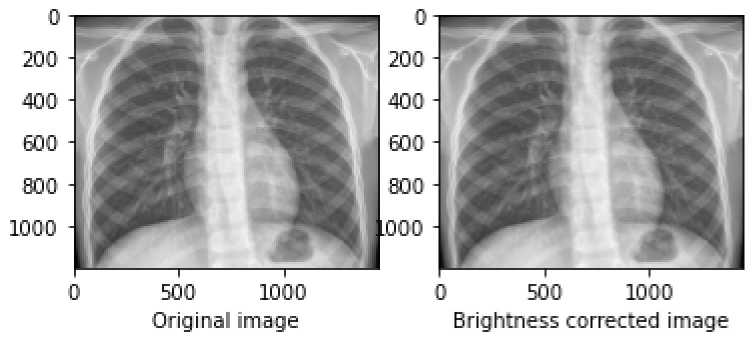
fig 1 and 2 : Chest x-ray images before (left and after (right) the cropping algorithm.





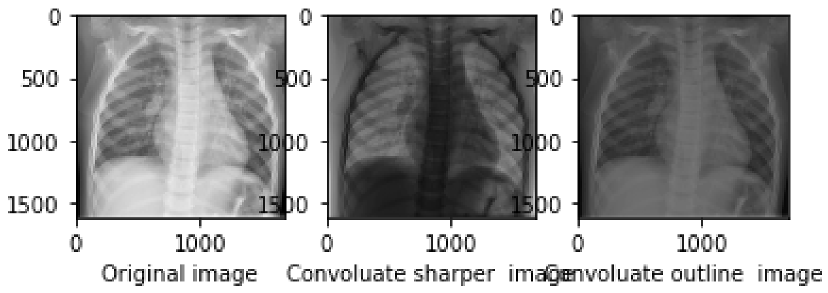
3.3) Correct the brightness

In order to reduce the differences between the images of our database, we implemented an algorithm to correct the brightness of each picture. This algorithm stretches all the pixels to ensure that the darkest pixel is of value 0 and the brightest of value 255. Unfortunately, applying this algorithm reduced our testing accuracy by 2% : we decided not to use it.



3.4) Convolute the images

We tried to use two filters on our images, one filter improving the outlines, the other the sharpness. Using them has not improved our testing accuracy.



3.5) Create our x_train, y_train, x_test and y_test

For each image, we create a tuple [pix, label] that we add to the list D. The label is 0 for normal images, and 1 for pneumonia images. We paid attention to have the same number of pneumonia and normal images because a bigger proportion of a certain class could affect our results. We proceed the same way for every picture of our database and we randomly shuffle D. Then we divide D in 4 lists to create our data. The first 90% of D will be used to create x_train and y_train. The last 10% to create x_test and y_test.

Finally we have a training dataset of 2318 images and a testing dataset of 258 images.

4) Exploring some estimators

Now that we have our data ready to be used, we need to implement some models and then we have to find the best model to predict if a patient has pneumonia.

4.1) Multilayer Perceptron (MLP)

The first model we tried to use is the multilayer perceptron.

- The home-made algorithm

We firstly tried to use the implementation we did in the lab2 to see if it would work on our data. We had to rearrange the algorithm a bit to make it work. With 50 epochs, a learning rate of 0.26 and 100 hidden layers, we obtained a 94% test accuracy.

- The MLP Classifier of Scikit learn

First we had to understand what the scikit learn classifier was doing and which parameters to give to it. Indeed, there are a lot of parameters that can be changed. The most important one are : hidden_layer_sizes(the number of hidden layers and the size of each hidden layer) , activation (activation function for the hidden layer), solver (the solver for weight optimization), alpha (regularization term) and learning_rate.

4.2) The Super Vector Machine (SVM) of Scikit learn

Then, we implemented a Support Vector Machines (SVM) model on scikit learn, here we also had to decide which parameters to use. Here the important parameters are : the regularization parameter C, the kernel type to be used in the algorithm (linear', 'poly', 'rbf', 'sigmoid', 'precomputed'), and gamma the kernel coefficient.

5) Model selection

Finding the best parameters for each estimator is not an easy task. Indeed, the number of parameters and the number of options per parameter is important and implementing the combination of all parameters is not possible. Thus, the first thing to do is to search in the documentation what parameters fit the best with our problem.

Hence, we learnt that for the MLP estimator, the default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score; for small datasets, however, 'lbfgs' can converge faster and perform better. So we focus more on the 'lbfs' parameters as our dataset is not so big.

Then, we did some "tests" on a validation set to see what parameters influenced the most and are the ones that need to be optimized. In order to have an historic of what we did, we created a document in which we save all the results we computed with the parameters we used.

Then, we implemented a cross validation algorithm to evaluate the estimator performance depending on some parameters. The principle is that we create a set of k combinations of parameters with different values that we want the estimator to perform (the set is usually composed of 5 different combinations). We divide the train set in k sets equal in size. For each combination of parameter values , we train the estimator on (k-1) sets and test on the set that did not serve for the training. Each set is used only once for testing (the train/test sets change for each combination). The estimator selected with the best combination parameters is the one with the better results on the test set, the final test results are given with the score made by the selected estimator on the test dataset.

Finally, to make sure the model selected does not work well only on a precise dataset, we wanted to test the robustness and to evaluate estimator performance. Thus, we also did cross validation with the *cross_val_score* scikit learn function. By running multiple times the estimator on different datasets, and by looking at the average accuracy and the standard deviation, it avoids the randomness of having a particular dataset and thus avoid overfitting.

6) Results

- **Home-made algorithm**

With our home-made algorithm we managed to have 94% accuracy on the testing set.

- **The MLP classifier of Scikit-learn**

At the end of the cross validation, the selected candidate is :
hidden_layer= 300, activation= 'logistic' and solver= 'lbfgs'.
With this estimator, we attempted to have 100% accuracy on the train dataset and 95% accuracy on the test dataset.
When we did the validation on the training set we had : 94% accuracy in average with a standard deviation of 1%, which means the model is quite reliable.

- **The SVM classifier of Scikit-learn**

At the end of the cross validation, the selected candidate is :
C=4000, kernel= 'rbf' and gamma=0.002
With this estimator, we attempted to have 100% accuracy on the train dataset and 95% accuracy on the test dataset.
When we did the validation on the training set we had : 94% accuracy in average with a standard deviation of 1%, which means the model is quite reliable.

7) Conclusion

For two estimators with the same accuracy, the best one is the more reliable one, which is the one with the least deviation on the validation score.
Here, we have the same deviation on both MLP and SVM, but the SVM accuracy is better (95,7%) thus choose this estimator.

One way to improve our results could be the use of a convolutional neural network (CNN). We implemented an algorithm using the keras library. Unfortunately, we have not been able to make it work properly by lack of time, and as we did not see this algorithm in the course, the understanding of how it works was difficult.

What we can learn from these results is that , on 100 patients, our algorithm predicts well 95 patients. It is good, but it might not be enough. It is important to avoid mistakes in the detection of pneumonia because it deals with the patient's health and an unrecognized pneumonia is a threat that can lead to the patient’s death. 5 out of 100 people are not correctly diagnosed is too much. Hence, our algorithm can be used as a decision helper but it cannot replace the judgment of a doctor.