COMS 4771 COVID Kaggle Competition report (Spring 2020)

Bertrand Thia-Thiong-Fat
Master student in Management Science and Engineering
Department of Operations Research
Columbia University
New York City, New York
Email: bt2513@columbia.edu

Given the present impact of the coronavirus on our lives, it made sense to work on some real-world data relevant to the current crisis. In this context, we worked on a classification challenge hosted on Kaggle [1] and presented during the COMS4771 Machine Learning course. The objective of this work is to diagnose patients with Covid-19, viral pneumonia, and bacterial pneumonia from images of chest X-rays. We will develop a multi- class classifier and aim a good weighted categorization accuracy on a set of unseen examples. In this paper, we will describe our approach to the problem and lessons learned during the process. Our final experimental results showed an accuracy of 82%.

1 Introduction

Early in the epidemic, physicians were diagnosing cases of coronavirus using X-Ray and CT images. It thus makes sense to work on developing more accurate methods for diagnosing Covid-19 from chest X-Rays. Since Covid X-Rays are frequently confused with ordinary pneumonia, our mission is to perform multi-class classification, distinguishing patients with Covid-19 from those who have viral and bacterial pneumonia or who are healthy [1].

1.1 Dataset

The training data (available on Kaggle) includes 1127 chest X-Rays drawn from several different sources (of varying size and quality) and a set of multi-class labels indicating whether each patient was healthy or diagnosed with bacterial pneumonia, viral pneumonia, or Covid-19.

1.2 Approach

With the recent advances in deep learning and the rise of edge computing, the field of computer vision has changed. Neural networks proved to deliver accurate results and be the state-of-the-art techniques for image recognition, as observed during many ImageNet competitions. We thus chose to implement such models in our study: our objective being to reach the highest performances possible, it only made sense to try to fit one of the best models existing to our problem. However, the main drawback of these algorithms is that they are difficult to interpret: we will thus not address this problem in this work.

That being said, there are many ways of implementing neural networks. We decided to use the API Keras to code our models. As described on its website [2], Keras is a high-level neural networks API, written in Python. It was developed with a focus on enabling fast experimentation, and allowed us to quickly build neural networks and processed our data during our study.

2 Exploratory Data Analysis

Before beginning the model building part of the project, we needed to better understand the data at hand and their characteristics. We thus started to load the chest X-Ray images from our training set to observe them. As mentioned previously, the training data is composed of 1127 samples. We observed 350 samples of normal diagnoses, 350 samples of viral pneumonia diagnoses, 350 samples of bacterial pneumonia diagnoses, and finally 77 samples of Covid-19 diagnoses. The first point to notice is thus that our dataset was imbalanced, and we would have to deal with this issue in our study. Moving on, we also saw that the chest X-rays were of different sizes and qualities, this heterogeneity representing another problem to address later. Reading the training data on Python, we obtained grayscale images. Figure 1 shows some samples of our dataset for different classes.

We can see that it is difficult to differentiate the different cases, hence the relevancy of implementing an accurate multi-class image classifier. Also, we can observe the heterogeneity of shapes and quality of our images, as mentioned previously.

In the end, this step allowed us to have insights about our data and highlighted two points to address: the imbalance problem and the heterogeneity of shapes and qualities.

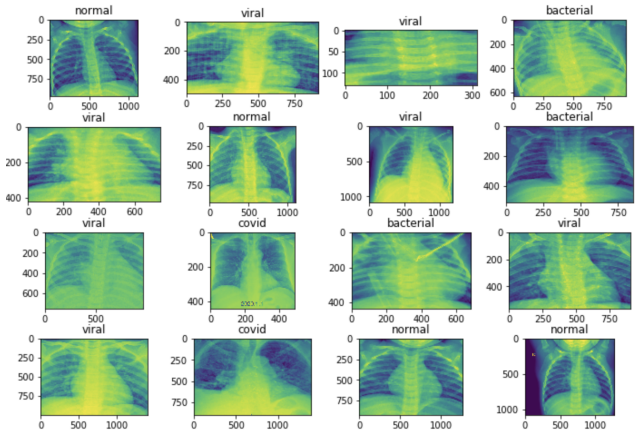

Fig. 1. Samples chest X-rays from our dataset and their class

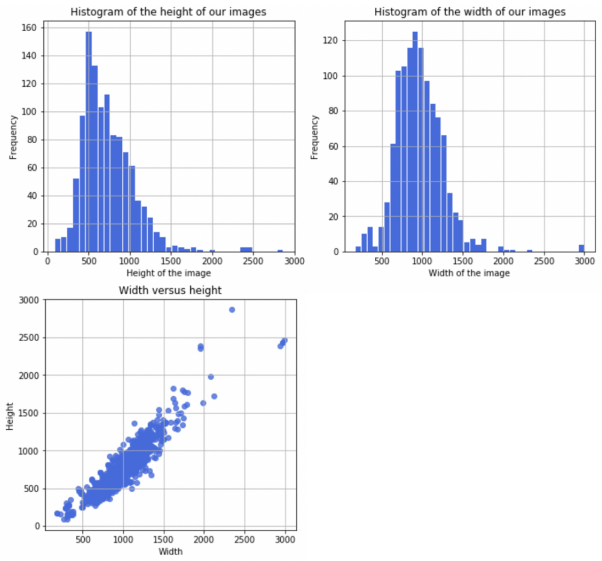

Fig. 2. Distribution and relationship of the width and height of the X-rays images

3 Data processing

To handle imbalanced datasets, two main techniques exist: class weighting and sampling methods. Since our dataset is small, we chose to only explore class weighting. This method consists in giving different weights to the examples when computing the loss during the training of a model. As a result, it becomes possible to penalize more the errors made on the minority classes, improving their classification that may be overlooked by majority classes otherwise. Since this step occurs during the training of one model, we will further discuss about it later. Coming back to the sampling methods, we chose not to explore them since they did not look well suited to our problem: our dataset is small and the risk of overfitting seemed important if we used undersampling; furthermore, this makes the latter sensible to noise and generating synthetic data did not seem the right way to go.

To improve the uniformity of our dataset, we started by resizing the chest X-rays. We first analyzed the relationship between the width and height of our images to find the optimal shape.

Figure 2 shows that this relationship is quite linear. We could thus expect that resizing our data would not distort them too much in an heterogeneous way, but uniformly. We therefore used the median dimension of our images (755 x 936) as point of reference in our study. We can see on Figure 3 the effect of the resizing: it made our data look more uniform.

Going further, [3] denoising images using a gaussian smoothing is popular in computer vision. We thus considered this technique and saved the original training data after application of a gaussian blur to remove potential noise. We will later discuss the efficiency of such treatment on the performances of our model. Though the effect of the denoising was not visible for all of the images, we noticed that it improved the quality of some of them (Figure 4). We could therefore assume similar improvement of X-ray images across our dataset.

Finally, we standardized the pixel values of the images to improve the uniformity across our dataset. Setting a mean of zero and a standard deviation of one also allowed us to scale the values, process that generally speeds up the learning and convergence of neural networks [4] due to the shape of the activation functions used.

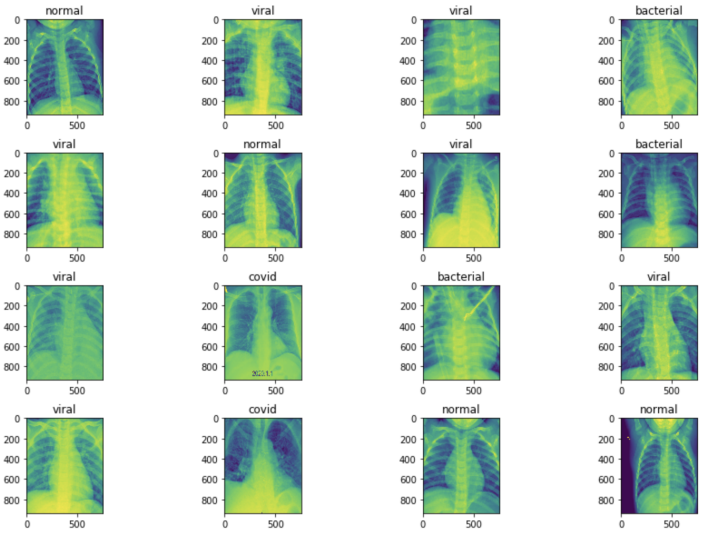

Fig. 3. Samples chest X-rays after resizing

4 Baseline model

The preprocessing of the data being done, we then worked at building a baseline model from scratch to see how far we could get with a simple algorithm. To assess the performances of our model, we thus splitted our training data into a smaller training set using 80% of our data, and the 20% left were used as a validation set - we also used a random seed to make sure that we could keep the same setting during our study. This allowed us to monitor the accuracy of our model during the training, and make sure that it did not overfit the data, losing its generalization power. This

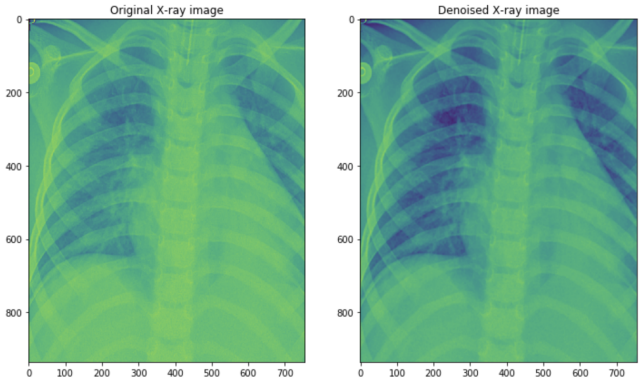

Fig. 4. Visible effect of the denoising treatment

operation was done using the 'train_test_split' function of sklearn, with the parameter 'stratify' to split the data and keep the same target distribution in both sets. This is important here since our dataset is small and imbalanced. To load the data and standardize them at the same time, we used the ImageDataGenerator object of Keras that allows to do it very conveniently, with the additional possibility of performing data augmentation and make the classification task more challenging. This feature can be very handful to build more robust models, but we chose not to explore this option yet for our baseline model, and rather see later how it could improve our results. Because our dataset is small, we decided to load it using a batch size of 32. The input shape of the convolutional neural networks used are generally of the order of 250 x 250 pixels. We thus decided to divide by 3 the dimension of reference found during our data exploration, resulting in images of size 251 x 312. Reducing the dimension of our images too much would lead to a loss of quality, while keeping it too high would make our models more complex and too long to train.

Our images ready to be used, the next step was the construction of our neural network. The architecture of such models are often described to be an art in some articles, as there is no clear rule stating how to build the best one, and it is mostly from intuition, luck and trials. Having limited knowledge in the classification of X-ray images, we did some research and finally built our architecture using inspiration from this article [5]. Since our dataset is small, a too large number of layers and parameters would just lead to overfitting. We thus decided to just implement two convolutional filter blocks - with reasonably low numbers of neurons - and made sure to use regularization to avoid the overfitting issue (with BatchNormalization and Dropout layers). For the classifier part, we chose to use only two layers with Dropout again. In the end, here is the architecture of our baseline model:

Before training our model, we used the 'compute_weight_class' function of sklearn to perform class weighting - as mentioned in the previous section - and calculated the weights allowing to consider each class equally despite their different population. We then compiled our model with the Adam optimizer (no specific reason, other that it

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 310, 249, 16)	160
max_pooling2d_1 (MaxPooling2	(None, 155, 124, 16)	0
conv2d_2 (Conv2D)	(None, 153, 122, 32)	4640
conv2d_3 (Conv2D)	(None, 151, 120, 32)	9248
batch_normalization_1 (Batch	(None, 151, 120, 32)	128
max_pooling2d_2 (MaxPooling2	(None, 75, 60, 32)	0
dropout_1 (Dropout)	(None, 75, 60, 32)	0
flatten_1 (Flatten)	(None, 144000)	0
dense_1 (Dense)	(None, 128)	18432128
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8256
dropout_3 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 4)	260

Fig. 5. Architecture of our baseline model

is quite popular) and trained our model on 50 epochs. The training curves can be seen below:

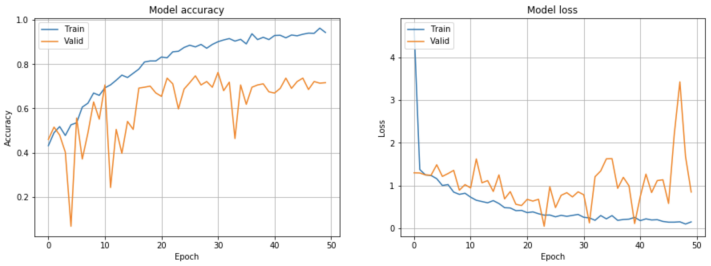

Fig. 6. Training curve of our baseline model

We can observe signs of overfitting starting approximately after 30 epochs, with an increase of the validation loss while the validation accuracy does not improve anymore. Nonetheless, we were able to obtain an accuracy of 73% with our baseline model with satisfying results on each class, as the classification report on Figure 7 shows.

	precision	recall	f1-score	support
0	0.64	0.71	0.68	70
1	0.85	0.69	0.76	16
2	0.86	0.94	0.90	70
3	0.66	0.54	0.59	70
accuracy			0.73	226
macro avg	0.75	0.72	0.73	226
weighted avg	0.73	0.73	0.73	226

Fig. 7. Classification report of our baseline model

We can see that our baseline model is capable of classifying Covid and normal cases quite well, but has some trou-

bles with the two types of pneumonia. Overall, the performances are still acceptable with this first model, and we will work at improving them.

5 Improving our baseline model

5.1 Assessing the impact of data augmentation

Our next work was the improvement of our baseline model. We kept the same setup for the training and validation sets, and loaded the data similarly as previously except that we used data augmentation this time. Indeed, since we previously saw that our model was overfitting quite quickly, we wanted to address this problem and see if, by using data augmentation, we could allow our model to learn more from the data, and be more robust to generalization at the same time. We thus chose to keep the same architecture as previously to assess the impacts of data augmentation exclusively. We initially augmented our data with rotations, zooms, shear and flipped images at the same time, but we quickly realized that using too many image transformations at the same time prevented our model to learn efficiently and the training accuracy could not improve. We assume this is because our dataset is small, and adding too many transformations may just bring noise because of the reduction in proportion of the original examples. We thus started to just add small rotations to our dataset to not disrupt it too much and saw that our network was able to learn well. The training curve can be found in appendix (it was too large to be put here). Though we trained our model longer, we can see that the effect of overfitting are diminished. We were also able to reach slightly higher performances and in a consistent way. This model gave us 78% of accuracy, thus an increase of 5% compared to our previous baseline model without data augmentation. Looking at the classification report (Figure 8), we can see net improvements for the classification of all of the classes in our validation set.

In conclusion, the use of data augmentation did improve the performances of our model and helped at addressing the overfitting issue. Though we saw that adding too many transformations was counterproductive and seemed to disrupt our dataset, reasonable augmentations proved to be effective in the end.

	precision	recall	f1-score	support
0	0.75	0.69	0.72	70
1	0.87	0.81	0.84	16
2	0.82	0.96	0.88	70
3	0.74	0.69	0.71	70
accuracy			0.78	226
macro avg	0.79	0.79	0.79	226
weighted avg	0.78	0.78	0.77	226

Fig. 8. Classification report of our baseline model

5.2 Assessing the denoising treatment as a preprocessing step

As mentioned in the Data Preprocessing section, we used a gaussian smoothing on our training data to denoise the X-ray images. In this section, we are going to discuss about the effectiveness of such preprocessing on our training data. We previously saw that it looked to improve some of our images (Figure 4). To assess the effects of the denoising, we kept the same settings as previously for the split of our training data, and used the same data augmentation as well (small rotations) - making sure to use the same random seed to obtain the exact same augmented dataset. We then trained our ConvNet on the denoised images saved previously. The training curves obtained are the following:

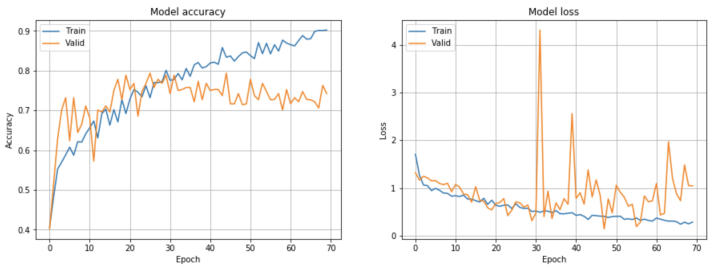

Fig. 9. Training curves for the denoised data

We can see effects of overfitting earlier this time, but our model was able to reach slightly better performances before that. Indeed, we managed to obtain better and aligned results between the training and the validation sets around the epoch 21 with 79.2% of accuracy. The classification report shows improvements for the classification of Covid and normal cases compared to previously:

	precision	recall	f1-score	support
0	0.74	0.70	0.72	70
1	0.88	0.94	0.91	16
2	0.94	0.90	0.92	70
3	0.68	0.74	0.71	70
accuracy			0.79	226
macro avg	0.81	0.82	0.82	226
weighted avg	0.80	0.79	0.79	226

Fig. 10. Classification report for the denoised data

Though it seems difficult to tell if these better results are because of luck with the stochastic processes involved in the training or real improvements due to a better preprocessing, after running the same model multiple times, we observed similar small improvements most of the time. In conclusion, denoising the data did slightly improved our results. The gains were not significant, but it seemed that it still helped our model a bit. We thus used the denoised data to train our model during the rest of our work.

6 Exploring transfer learning

Based on our work so far, we tried different settings of data augmentation and experimented several architectures of neural network to improve our results. We previously saw that adding too many transformations to our dataset was not effective, so we used one transformation at a time and explored the zoomed, flipped, sheared and shifted augmentations. We then modified the architecture of our network: when there were too many layers and parameters, the model indeed could not learn effectively while not enough parameters led to overfitting. In the end, despite our numerous attempts, we did not manage to get better results than previously. We thus decided to explore transfer learning and pre-trained models. The description of many models can be found on the Keras website [5] and this library allows to implement them quickly [6].

6.1 VGG16 model

As mentioned above, constructing ConvNets with too many layers seemed to be ineffective and not give good results. We thus wanted to start our exploration with an architecture not too deep and chose the VGG16 model. We kept the same settings to split our training data, and only used rotations to augment our dataset - since we previously that it was the most effective transformation for us. We then loaded the weights of VGG16 obtained from the 'ImageNet' dataset and only trained the classifier part of the network that we designed ourselves. We actually constructed the classifier identically as the one from our baseline model in the last section. Though there is no X-ray images in the ImageNet dataset, we decided to use it since we found it could still allow to build strong classifier of X-rays [7]. In the end, just training the classifier part of the ConvNet yielded 81% of accuracy. We can see on the classification report (Figure 11) that this model classifies better the pneumonia cases than our previous model while the performances on the Covid and normal cases are quite similar.

	precision	recall	f1-score	support
0	0.82	0.71	0.76	70
1	1.00	0.81	0.90	16
2	0.84	1.00	0.92	70
3	0.75	0.74	0.74	68
accuracy			0.82	224
macro avg	0.85	0.82	0.83	224
weighted avg	0.82	0.82	0.81	224

Fig. 11. Classification report for the VGG16 model

The results looking better than previously, we fine-tuned the network by training the last convolutional block of VGG16 as advised in the article [8]. It indicates that training some layers while leaving others frozen for problems with small datasets and that are different from the pre-trained model's dataset is generally a good way to get better performances. We thus froze all of the layers of the VGG16 network except its last block and trained it with a slow learning rate as advised in [6], while the weights of the classifier part were initialized with the ones obtained during the previous training. We finally obtained excellent performances a validation accuracy consistently around 89%.

6.2 Exploring deeper network architectures: ResNet50 and Xception models

Even though too deep architectures did not give good results during our experiments, we wanted to see the performances given by existing and popular ConvNets such as ResNet50 and Xception [7]. Proceeding similarly as with VGG16, to begin, we only trained the classifier part of the networks over 50 epochs. After some experiments, we observed that the models were overfitting very quickly and less than 30 epochs were enough to reach the best performances for both of them - probably due to their deep architecture. Also, using the same structure as previously for the classifier part gave worse results. We thus tried different structures and conducted some experiments on both models: we decreased the number of layers and neurons when we observed overfitting and used regularization, and increased them otherwise. In the end, the best accuracy we obtained for ResNet50 was 77%, and for Xception 72% after our trials. Since these structures are very deep and we did not obtain better results than with the VGG16 network, we decided not to dive further and fine-tune them.

6.3 DenseNet121 model

After reading the article [10], we tried to use the pre-trained DenseNet121 model for our task. The article describes the architecture of the network and explains how its design can help addressing the depth issue of many ConvNets. Because our problem seemed very sensitive to this point based on our previous experiments, we wanted to assess how DenseNet121 would handle the classification. We thus trained the classifier part of the network - constructed similarly to our VGG16 - and just added horizontal flip augmentations because the network is deep and would be able to handle them. The training curves can be found in appendix. The performances of our model were good and we managed to obtain an accuracy of 80%, close to the one we got with the VGG16 model (being our best result so far). The classification report below shows higher performances on the Covid and normal cases than our VGG16 model, but lower performances for the cases of pneumonia - hence a slightly lower accuracy overall than the latter:

We can indeed see that our model was capable of identifying 100% of the Covid cases in our validation set while capturing 94% of them. Because on the Kaggle website the classification of Covid cases has a bigger weight than the other classes, this model gave us the best score on the leaderboard (with 82.7%).

Excited by this result, we went further and fine-tuned the last bloc of the DenseNet121 model. We thus initialized the weights of the classifier part of our model with the ones obtained previously, and then froze all of the layers except the

	precision	recall	f1-score	support
0	0.73	0.66	0.69	70
1	1.00	0.94	0.97	16
2	0.95	0.99	0.97	70
3	0.67	0.72	0.70	68
accuracy			0.80	224
macro avg	0.84	0.83	0.83	224
weighted avg	0.80	0.80	0.80	224

Fig. 12. Classification report for the DenseNet121 model

ones of the last block of the DenseNet121 network before training our model with a slow learning rate. After 5 epochs only, the training showed excellent performances with accuracies of 87%.

7 Summary of the results

At the end of the day, please find below a table recapitulating our results along our study:

Model	Accuracy (%)
Baseline	73.0
Baseline 1	77.9
Baseline 2	79.2
VGG16	81.7
ResNet50	77.2
Xception	72.3
DenseNet121	80.0

Table 1. Final results table

Baseline 1 corresponds to the results obtained with our baseline model using data augmentation (small rotations only). Baseline 2 corresponds to the results obtained with our baseline model using data augmentation (small rotations only) and trained on the denoised data. All our pre-trained models were trained on the denoised data too and on the same examples (random seed set to 0 when splitting our training set and for the augmentations for all of our experiments).

8 Conclusion

This project was the opportunity to learn more about the Computer Vision field. Working on the multi-class classification of chest X-rays allowed us to get a better understanding of the preprocessing of images and the implementation of Convolutional Neuron Networks. We also handled an imbalanced dataset and got more familiar with transfer learning. Being able to manipulate popular network architectures such as VGG16 and ResNet on a concrete project was truly interesting. In the end, we applied and explored new algorithms, and finally managed to obtain accuracies in the order of 80% which was very rewarding.

9 Discussions

If one wanted to use one of our models in a clinical setting, considering the situation we are living, the DenseNet121 model would be the most relevant one since it has the highest performance for the detection of Covid cases. Its classification report indeed indicates a precision and recall close to 100% meaning that the model is highly effective at detecting nearly all Covid cases and with an almost perfect level of confidence. Figure 13 shows the confusion matrix of our model on our validation set. The classes 0,1, 2 and 3 represent respectively bacterial pneumonia, Covid, normal and viral pneumonia cases. We can thus see that among the 16 Covid cases, our model was able to identify 15 of them as such and just made one error. The classification of normal cases is also almost perfect. However, we can see that our model has trouble with the two types of pneumonia present in our data and tends to confound the two. If one used our model to diagnose patients, the errors on the two types of pneumonia would be large on a big population, though the identification of Covid and normal cases would be relatively low. As a result, building two separate models, one tailored for the classification of normal and Covid cases, and one tailored for the classification of the two types of pneumonia may be a good solution to drive more accurate diagnoses.

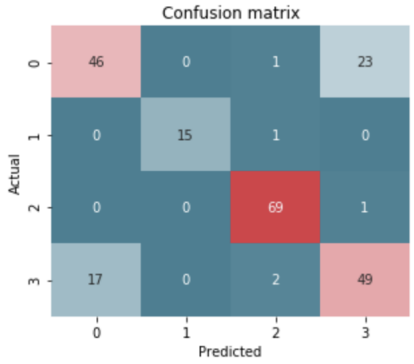

Fig. 13. Confusion matrix of the DenseNet121 model on our validation set

To drive better performances at the classification task, one could try to use Ensembling methods and average the probability results of several different models to obtain more accurate results. We did not have time to explore this option and thought about it too late to be honest.

References

[1] https://keras.io

[2] https://www.kaggle.com/c/4771-sp20-covid
[3] Towardsdatascience, *Image pre-processing*
[4] Towardsdatascience, *Why-data-should-be-normalized-before-training-a-neural-network*
[5] Towardsdatascience, *Deep-learning-for-detecting-pneumonia-from-x-ray-images*
[6] https://keras.io/api/applications/
[7] keras.io, *Building-powerful-image-classification-models-using-very-little-data*
[8] Towardsdatascience, *Does-imagenet-pretraining-work-for-chest-radiography-images*
[9] Towardsdatascience, *Transfer-learning-from-pre-trained-models*
[10] Towardsdatascience, *Understanding-and-visualizing-densenets*

Appendix

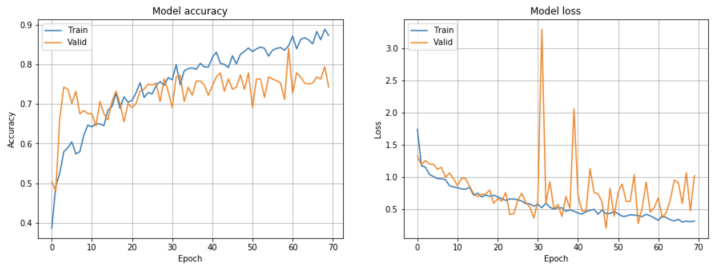

Fig. 14. Training curve of our baseline model with data augmentation

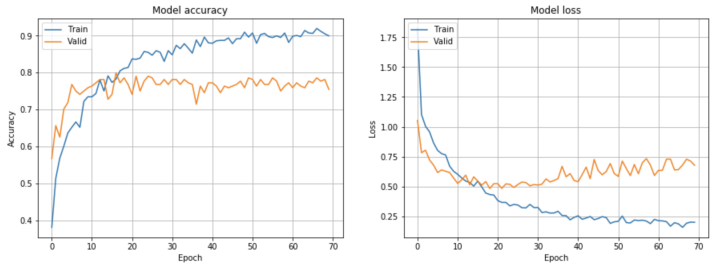

Fig. 15. Training curve of our DenseNet121 model