Classification of Chest X-Ray for COVID-19

Marco Lorenzo Piretto[e20201805], Gabriele Morina[e20201804], and Marius A. Hessenthaler[e20201824]

NOVA IMS, Lisbon, Portugal

1 Introduction

1.1 Motivation

With the ongoing pandemic of the novel and highly contagious SARS-CoV-2 and its high hospitalization rate reliable diagnostic tests for the COVID-19 infection (Covid) are of high importance in hospitals. The current gold standard for detecting Covid is the Polymerase chain reaction (PCR) test. While PCR tests have long analysis time, antigenic Covid tests are faster but also less reliable. In hospitals with the necessary medical equipment computed tomography (CT) and radiography (X-ray) scans create a possibility of fast and reliable detection of a Covid infection. As well as diagnose more progressed infections that do not yield high enough viral loads for throat or nasal swaps to be detected [1].

1.2 Goal

We want to train a classifier that can detect whether a patient has Covid or not. The classifier should not only detect Covid from healthy person but also differentiate Covid from other respiratory infection because it needs different treatment and isolation. The classifier will receive an chest scan image with at least 150 x 150 pixels and output either positive or negative. Our aim is to create a Convolutional Neural Network that is able to detect the cloud-like structures in the lung like seen in Fig. 1 with the goal to optimize both precision and recall.

1.3 Dataset

For training and testing the classifier we use the "DeepCovid" data set [3] which can be found at https://www.dropbox.com/s/9w8nmj791c9ogsx/data_upload_v3.zip. The training set is composed by 2084 images, 84 of patients who had Covid, while the remaining 2000 belong to either healthy people or patients with other pathologies (like edema, fracture, lung lesion). The test set is similarly composed, with 100 Covid images and 3000 non Covid images. One of the main difficulties of our project has been to deal with such an unbalanced dataset, this aspects are discussed in the Sections 2 and 3. In Figure 1 and 2 we provide examples of a positive and negative Covid chest X-ray.

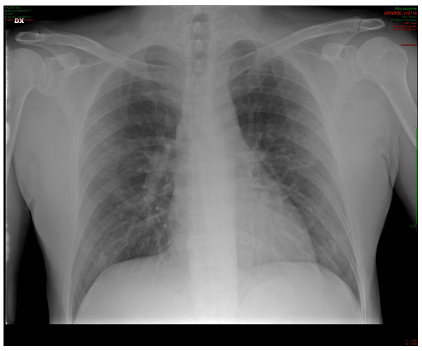

Fig. 1. Chest X-ray of a patient with Covid.

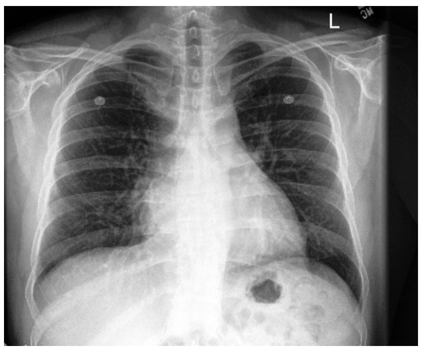

Fig. 2. Chest X-ray of a patient without Covid.

2 Pre-Processing

With our dataset splits ready, we need to format the images into appropriately preprocessed floating point tensors for feeding them into the Convolutional Neural Network. For this we used the ImageDataGenerator class from Keras that allowed us to create different generators for the validation and the training sets. Moreover all the images were converted from RGB to grayscale and resized to 150x150 pixels.

2.1 Data Augmentation

Data augmentation is an important technique that is particularly useful to increase generalization in our case given the small (especially towards Covid) data set at hand. In particular, looking at the images that we are dealing with, the augmentation that we decided to implement are: *horizontal and vertical shifts* both in a small range of [-0.1,0.1] in order to do not move the lungs out of the picture, *horizontal flip*, *random rotation* with a range up to 20 degrees, *random brightness* in a range of [0.8,1.2] in order to prevent that the "clouds" in the lungs that we are looking for are not visible anymore and *random zoom* in a range of [0.8, 1.3] in such a way that the lungs are not too small to be properly analyzed and are still clearly visible.

2.2 Oversampling

The training set is highly unbalanced with 2000 images in the negative class compared to only 84 images in the positive class. To deal with this issue, we considered either using the built-in "class weights" argument of keras or oversampling the minority class, that is the positive Covid class. With the advantage of data augmentation which will replace every duplicated image by an augmented image, we decided to use the oversampling approach.

3 Validation

3.1 Metrics

Because of the unbalanced class distribution and the special interest in the Covid class we will use the f1-score with Covid as the positive to evaluate the model in the optimization phase but also to finally evaluate our model on the test set. For the validation we keep track of the accuracy, precision and recall. In the course of the optimization we noticed that recall is often much lower than precision however for the problem more important therefore we put special attention to it.

3.2 Validation Split

The data set is very small especially in the Covid class. For that reason we decided to use 10-fold cross-validation to get more reliable estimates of the different architecture and hyper-parameters settings. Because of the unbalanced data set we use stratified sampling to ensure an even distribution of classes among the splits. For each fold we plot all the metrics, in order to detect overfitting or anomalies. In order to evaluate a classifier we first save with callbacks the weights for all epochs. For each fold, we choose the best epoch in terms of f1-score, selecting the earliest one in case of ties. Then we choose as validation epoch the median of the best epochs and perform the prediction at this step. We use these epoch's weights to predict the label for the corresponding validation data to calculate the overall f1-score. In order to ensure that the results are consistent we repeat the process for the adjacent epochs.

4 Architecture and Parameter Optimization

The get a first idea of well-performing network architectures we tested an initial architecture seen as Model 1 in Table 1 which achieved a f1-score of 0.84.

4.1 Optimization Plan

From the already well performing initial model we tried out if less or more complex architectures would increase the model performance. First we tried to change the architecture of the convolutional layers and also increased the dropout and the number of dense layers to fit the convolutional layer complexity which are summarized in Table 1. For the less complex architecture we achieved a f1-score of 0.86 and 0.89 for the more complex. After finding two well performing architectures we turned to fine-tuning the hyperparamters for both architecture complexities which are summarized in the Tables 2 for the less complex architecture (Model 2) and 3 for the more complex architecture (Model 6).

4.2 Convolutional Layers

In the direction of less complex architecture we tried to remove the first layer with 32 filters (Model 3). This change though did not bring any improvement, so we decided then to keep the topology of the three convolutional layers fixed.
In the direction of more complex architecture we increased both the number and size of convolutional layers. The best performing model had five convolutional layers paired with 0.4 dropout (See Model 6 in Table 1). After adding first a 256 filters (Model 7) layer and then 512 filters layer (Model 8) the model performance decreased to 0.86 for the former and 0.83 for the latter in f1-score. While using constant filter size of (3,3) throughout all models we finally tested an increase filter size of (5,5) which decreased the model's f1-score from 0.98 (Model 2.8) to 0.91 (Model 2.9).

4.3 Pooling

After each convolutional layer we added one pooling layer to the model with a window of 2x2 which resulted in non-overlapping pooling. First we tried different architectures with max-pooling. While optimizing both final architectures we also tried average pooling which increased the f1-score for the less complex from 0.87 to 0.96 and the more complex model from 0.89 to 0.91. We hypothesize that average pooling increases the performance because for the detection of the cloud-like structures in the lung a filter does not detect single features but rather the strong and frequent appearance of them.

4.4 Dropout

The usage of image augmentation increases the risk of overfitting. To mitigate overfitting we tried different settings of dropout layers. With simple models we started with a final dropout layer of 0.2 while for the more complex models with a final dropout layer of 0.4 which increased the model performance from 0.85 (Model 4) to 0.88 (Model 5). Also for the simple model increasing the dropout rate to 0.4 led to better results (e.g. from Model 2.3 to 2.4). In particular from the plots of recall and f1–score we saw more stable results for the validation set across the different folds.

4.5 Dense Layers

The application of an additional dense layer of 256 increased the complex model performance from 0.88 (Model 5) to 0.89 (Model 6). After already applying average pooling instead of max pooling the f1-score was again improved from 0.91 (Model 6.4) to 0.92 (Model 6.5) by using three 128 unit layers instead of three 256 unit layers.
In the direction of a simpler model we removed one dense layer (leaving only one hidden dense layer of size 256). This change brought more stable results. We also tried to add regularization: we decided to use $L2$ regularization, that should penalize evenly all the weights. We found that adding regularization (Models 2.1-2.2) helps to obtain more stable results. Finally we tried modify slightly the topology: keeping fixed the number of neurons (256) we tried to evenly split them into two layers of size 128 each (Model 2.6). This modification actually improved the score to 0.92 and looking the plots across the folds the results are quite consistent.

4.6 Class Weights

In real applications it is arguable whether it is more beneficial to have more false positive or false negative classifications. In almost all model architectures the model precision was close to 1.0 while the recall was in the area of 0.8. To achieve a more balanced precision and recall performance which is relevant in real world applications but also to increase the model's f1-score we implemented class weights. Additionally to the oversampling of the positive Covid class to match the negative Covid class we weighted the positive Covid class more than the negative. On the less complex model we tested weighting the positive class two times and four times more than the negative class. While the four times weighting increased both the recall and f1-score (Model 2.1), the two times weighting did not (Model 2.3).

4.7 Optimizer

We started training the models with the adaptive moment (Adam) optimizer which yielded already good results. After trying the Root Mean Square Propogation (RM-SPROP) optimizer once we dismissed it because it performed 0.1 worse in f1-score than Adam.

4.8 Loss Function

As described in the Section 3.1 we use the f1-score to evaluate our model. However, it is not possible to choose the f1-score as the the loss function because it is not differentiable. For this reason we started with the standard loss function binary cross entropy (BCE) which still delivered very good results. When comparing the training and validation loss function plots we noticed that for almost all epochs consistently in all folds and models the validation loss was lower than the training loss. Combined with the knowledge of consistently larger precision than recall from Section 4.6, we had the hypothesis that the models were simply better at classifying negative Covid images than positive Covid images. With the oversampling (Section 2.2) we have an equal number of positive and negative images in the training set however in the validation set there are 95 % negative images which benefits the validation loss with our previous hypothesis.
To explore this observation further and optimize the f1-score further we tested an adapted version of the f1-score as the validation loss to allow gradient descent but enable better optimization for the f1-score. We used [2] implementation of "macro double soft f1-score" (MDS-f1) that uses probabilities instead of binary values to compute the f1-score for both positive and negative classes. However the resulting model performance was with 0.80 worse than BCE loss with 0.89 in f1-score. In the training and validation loss function plots we could see that the validation loss stagnated between epoch five and ten while the training loss further decreased (See Fig. 4), which we saw as a sign of overfitting.

4.9 Epochs & Overfitting

We started with 40 training epochs however the training and validation loss could not be increased after training more than 20 epochs (at least for the less complex models). Therefore we chose to reduce the number of epochs to 30. We used callbacks to saved the models weight after each epoch to retrieve the model for final validation. We used the median of the best epochs of each of the single KFolds iterations as the epoch to load the weights from for all models. Often the best values were around epoch 15: indeed, even if in the latest epochs there are spikes of very good scores, we also see that these scores are not stable and consistent across different folds. This way it was not clear to see whether the model was overfitting or not. However, we decided to further explore the matter.

As described in the previous Section, due to the oversampling in the training data and the underrepresented positive samples in the validation data the validation loss and the training loss with the BCE loss function are not very comparable. However, we can see that after approximately 20 epochs the validation loss is slightly increasing while the training loss still decreases (see Fig. 3). When looking at the f1-score training and validation plots we only see slight overfitting in some complexer models after approximately 20 epochs but mostly oscillation of the validation f1-score due to the small size of the validation set (see Fig. 5). We account the small and delayed overfitting to the measures of extensive data augmentation, rather simple model architecture, dropout layer and partially weight regularization which allows the model to generalize well which we could also observe when testing the model on unseen data (Section 5.2). Especially, when using the MDS-f1 as a loss function we could see overfitting through the stagnation of validation loss after approximately five epochs (See Fig. 4 and Section 4.8).

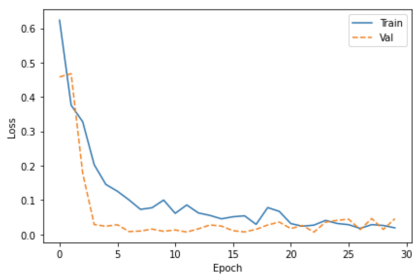

Fig. 3. Plot of the validation and training BCE loss by epoch.

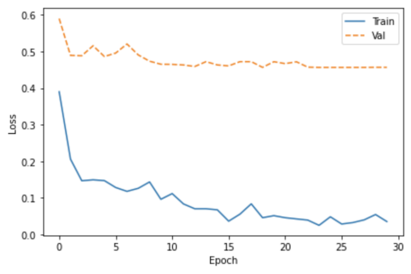

Fig. 4. Plot of the validation and training MDS-f1 by epoch.

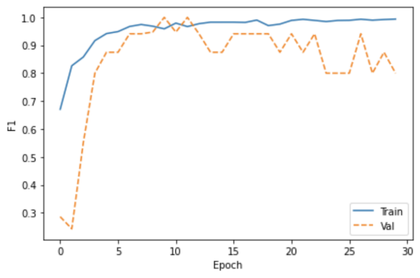

Fig. 5. Plot of the validation and training f1-score by epoch.

Table 1. Experiment results for the Architecture Optimization.

Nr.	Convolutional Layers *	Dropout Layer**	Dense Layers (***)	f1-score
1	32,64,128	0.2	256,256	0.84
2	32,64,128	0.2	256	**0.87**
3	64,128	0.2	256	0.86
4	16,32,64,128,256	0.2	256,256	0.85
5	16,32,64,128,256	0.4	256,256	0.88
6	16,32,64,128,256	0.4	256,256,256	**0.89**
7	16,32, 64,128,256,256	0.4	256,256,256	0.86
8	32, 64,128,256,256,512	0.4	256,256,256	0.83

(*) with filter size (3,3) and Max-Pooling of size (2,2) and ReLu activation funtion. (**) after the last Convolutional layer. (***) With ReLu besides the final layer with Sigmoid activation function.

Table 2. Experiment results for the Optimization of Model 2 from Table 1.

Nr.	Filter Size	Pooling	Dropout Layer	Dense Layers	l2 Weight Reg.	Class Weights	f1-score
2.1	(3,3)	Max. (2,2)	0.2	256	0.005	neg.: 0.5, pos.: 2	0.89
2.2	(3,3)	Max (2,2)	0.2	256	0.01	neg.: 0.5, pos.: 2	0.90
2.3	(3,3)	Max (2,2)	0.2	256	0.01	neg.: 1, pos.: 1.5	0.87
2.4	(3,3)	Max (2,2)	0.4	256	-	neg.: 0.5, pos.: 2	0.95
2.5	(3,3)	Max (2,2)	0.4	256	0.005	neg.: 0.5, pos.: 2	0.90
2.6	(3,3)	Max (2,2)	0.2	128, 128	-	neg.: 0.5, pos.: 2	0.92
2.7	(3,3)	Avg. (2,2)	0.2	256	-	neg.: 0.5, pos.: 2	0.96
2.8	(3,3)	Avg. (2,2)	0.4	128, 128	0.005	neg.: 0.5, pos.: 2	**0.98**
2.9	(5,5)	Avg. (2,2)	0.4	128, 128	0.005	neg.: 0.5, pos.: 2	0.91

Table 3. Experiment results for the Optimization of Model 6 from Table 1.

Nr.	Pooling	Dense Layers	l2 Weight Reg.	Class Weights	Optimizer	Loss-function	f1-score
6.1	Max (2,2)	256,256,256	–	–	RMSPROP	BCE	0.79
6.2	Max (2,2)	256,256,256	–	neg.: 0.5, pos.: 2	Adam	BCE	0.87
6.3	Avg. (2,2)	256,256,256	–	–	Adam	BCE	0.90
6.4	Avg. (2,2)	256,256,256	0.05	–	Adam	BCE	0.88
6.4	Avg. (2,2)	256,256,256	–	–	Adam	MDS-f1	0.80
6.5	Avg. (2,2)	128,128,128	–	–	Adam	BCE	**0.92**

5 Conclusion

5.1 Final Model

Even though the more complex models have similar performance as the simpler models we followed Ocam's razor and choose the simple model as our final model. Also we did not only took into account the single highest f1 score (that since the small sample size could also be due to a statistical fluctuation) but also consistency and stability of the results across epochs. Our final model is trained for 16 epochs and consists of:

– Conv2D with 32 filters (3,3)
– Average Pooling (2,2)
– Conv2D with 64 filters (3,3)
– Average Pooling (2,2)
– Conv2D with 128 filters (3,3)
– Average Pooling (2,2)
– Flatten layer
– Dropout 0.4
– Dense 128 (L2reg = 0.005)
– Dense 128 (L2reg = 0.005)
– Dense 1 (Sigmoid)

Besides the output layer, all layers have ReLu activation function.

5.2 Test Result

We only used the test set as an final evaluation to have an accurate estimate of the model's real life performance. So we built a new model with the same topology, and we trained it on all the train data. On the test set the final model achieved the following results.

Confusion Matrix:

		True diagnosis		
		Positive	Negative	Total
Predicted diagnosis	Positive	97	5	102
	Negative	3	2995	2998
	Total	100	3000	3100

Classification Report:

	precision	recall	f1-score	support
Negative	1.00	1.00	1.00	3000
Positive	0.95	0.97	0.96	100
Accuracy			1.0	3100

The model deviates only 0.02 in f1-score of from the model tested on the validation set. With the f1-score of 0.96 the model performs really well, able to generalize well and suitable for its intended task.

5.3 Error Analysis

Having obtained this final result, we decided to further investigate the images that the model is misclassifying. We observed that some of them have actually very poor image quality (See Section error analysis in the appended notebook) and the lungs are hard to see properly. Interestingly, we also note that out of the three false negatives, two belong to the same person. In conclusion, even if our architecture does not achieve 100% accuracy, misclassification of some images is likely due to their poor quality.

5.4 Discussion & Comparison

The best performing model constructed by the authors who prepared the dataset that we used [3] has similar results to ours. Indeed they were able to achieve a recall rate of 98%, almost the same with respect to the 97% that we obtained. As a second metric they used the in testing frequently-used measure of specificity and scored a maximum 92.9 %. Compared to that our model achieved an even better score of 99.8 %. The comparison is especially interesting for the fact that they used the approach of transfer learning with much more complex models compared to our approach that learned simple models from scratch. Anyway, such good results are probably mostly possible due to the fact that they put only those images with a clear sign of Covid in the data set. However it is worth to mention that to have a better generalization of the model and estimate of its performance, more Covid images would be needed.

References

1. Islam, H., Rahman, A., Masud, J., Shweta, D., Araf, Y., Ullah, M., Al Sium, S.M., Sarkar, B.: A generalized overview of sars-cov-2: Where does the current knowledge stand? European Journal of General Medicine **17** (05 2020). https://doi.org/10.29333/ejgm/8258
2. Maiza, A.: The unknown benefits of using a soft-f1 loss in classification systems (12 2019), https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d
3. Minaee, S., Kafieh, R., Sonka, M., Yazdani, S., Jamalipour Soufi, G.: Deep-covid: Predicting covid-19 from chest x-ray images using deep transfer learning. Medical Image Analysis (2020). https://doi.org/https://doi.org/10.1016/j.media.2020.101794