CNN Model for X-Ray Classification

Ana Luís, 20210671

Carolina Machado, 20210676

Francisco Calha, 20210673

Luís Santos, 20210694

Sara Arana, 20210672

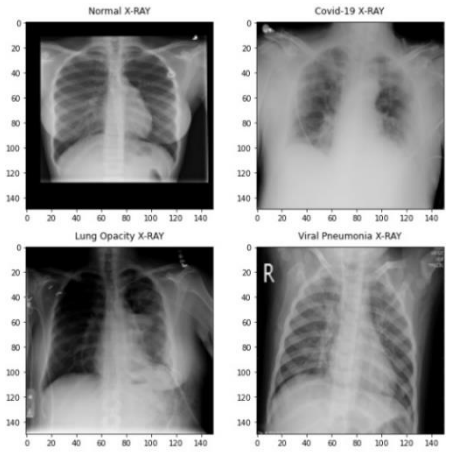
Introduction

For the scope of this project, we decided to explore the several practical applications of Deep Learning. From here, we discovered the impact of these algorithms in the healthcare sector. As such, we decided to choose a topic related to this area – Deep Learning usage in interpreting X-Rays.

Task definition

Our dataset was retrieved from Kaggle, and it consists of 21 165 lung x-ray images – which are our inputs. The outputs refer to the classes to which each x-ray is associated to – namely, Normal, Covid, Lung Opacity, and Viral Pneumonia. Our goal is to build a model that correctly classifies the x-ray it receives, by assigning it to one of these 4 categories. We found this problem very interesting because it might lead to more efficient functioning of emergency rooms in hospital, by possibly decreasing the patient's waiting time, and reducing some pressure off from medical staff.

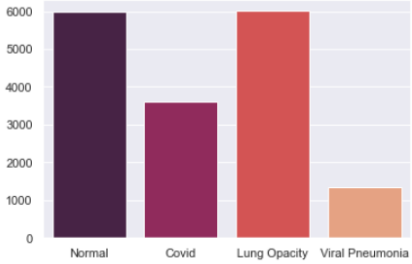
Fig. 1 - X-rays examples from each category



Our Dataset

Our dataset consisted of 4 folders, each one corresponding to one class. However, we were dealing with a very large dataset composed by 21 165 images – 10 192 for Normal, 3 616 for Covid, 6 012 for Lung Opacity, and 1 345 for Viral Pneumonia x-rays. Since our computational resources are limited, we decided to resort to under sampling. Our approach consisted on importing less images of the 'Normal' category, to also solve for possible problems from an imbalanced dataset. We ended up with a total of 16 974 images.

Fig. 2 – Countplot for the final dataset



To retrieve the data, we used the glob library to access our images path, and then used the load_img utility from Keras to load them. In the last step, we defined a maximum number of Normal x-rays, to decrease the discrepancy between this category the others. Afterwards, we transformed our images to array (img_to_array), in order to be able to work with them. Then we proceeded to split our dataset into Training, Validation, and Test, corresponding to 12 220, 3 056 and 1 698 images, respectively. By using the train_test_split, we were able to define stratify=targets, which allows us to have in each dataset partition, the same percentage of images of each class that the total dataset has.

Evaluation measures

To measure our model, we used two different metrics. The one we focused on the most was the Accuracy, but we decided to pay special attention to the Recall. These two measures allow us to see how well our model is learning and provide us prospects on what could be done to improve it.

Approach and Error Analysis

To build our model, we decided to explore image augmentation techniques in our training dataset, to possibly improve the outcome and performance of our models.

There were some parameters that we decided to keep equal in all our different approaches to the problem. First, we defined the input shape as (150, 150, 3) in all our models. We also tried to run one model for a (299, 299, 3) input shape, since it is the actual size of our images, however, the increase in performance did not outweigh the computational costs. Afterwards we defined the batch size at 32, since it is a common number and the results obtained were good, so we kept this parameter. We also run all our models for 32 epochs, however, we defined the early stopping class, to stop the model from continue running, after there is no loss decrease. This helped us reduce computational resources and time. We also used the ‘relu’ activation function in all convolution and dense layers, except in the last dense layer, in which we used the ‘softmax’ activation function, since we are dealing with a multi-class classification problem. For the loss function, we tried several variants of the ‘Sparse’ categorical one – we chose ‘Sparse’ because of the way our target variable was defined.

Notice that, for the following models, the ‘adam’ optimizer was used because it led to better results, however, the ‘rmsprop’ optimizer was also explored.

Model 1

We started by running a fairly simple model, using the ‘sparse_categorical_crossentropy’ loss function. The model was the following:

Fig. 3 – Model 1: Summary

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 148, 148, 32)      896

 max_pooling2d_2 (MaxPooling  (None, 74, 74, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 72, 72, 32)        9248

 max_pooling2d_3 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 flatten_1 (Flatten)         (None, 41472)             0

 dense_2 (Dense)             (None, 128)               5308544

 dense_3 (Dense)             (None, 4)                 516

=================================================================
Total params: 5,319,204
Trainable params: 5,319,204
Non-trainable params: 0
_________________________________________________________________
```

This model led to a training accuracy of 77.9% and validation accuracy of 79.6%. From the Training and Validation accuracy and loss plots below, we are able to see our model is slightly underfitted. This means that we could, possibly, add more complexity to our model (by adding more layers) and see if it starts to learn more complex patterns in the x-ray images.

Fig. 4 – Model 1: Training and Validation Accuracy

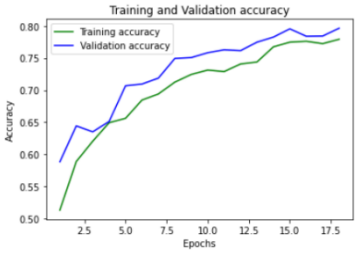
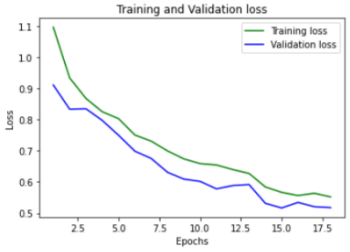


Fig. 5 – Model 1: Training and Validation Loss



From the classification report and the confusion matrix of the validation set, we were able to see the recall metric which showed us that this model was not predicting the images belonging to the Covid class as well as the other classes. This might be happening due to the possibility of the Covid X-ray detection being more complex, and for which we would need a more complex model. Another possible reason would be the fact that this class had less observations than the 'Normal' and 'Lung Opacity' categories. In the next model, we will try to solve this problem.

Fig. 6 – Model 1: Confusion Matrix

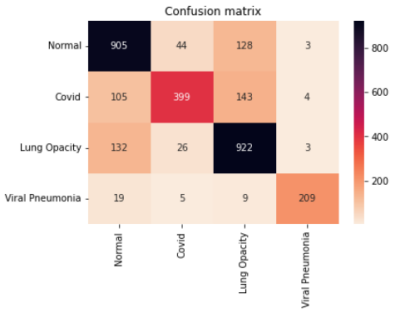


Fig. 7 – Model 1: Classification Report

	precision	recall	f1-score	support
normal	0.78	0.84	0.81	1080
covid	0.84	0.61	0.71	651
lung_opacity	0.77	0.85	0.81	1083
viral_pneumonia	0.95	0.86	0.91	242
accuracy			0.80	3056
macro avg	0.84	0.79	0.81	3056
weighted avg	0.80	0.80	0.79	3056

Finally, we also computed the accuracy for the Test set, which was 78.2% and the results from the confusion matrix and the classification report were similar to the ones shown previously.

Model 2

For our 2nd model, the classifier's layers were the same. The only changes we made were the loss function and the class weights. For the loss function, we used the 'SparseCategoricalFocalLoss' function which aims to help imbalance classes, by focusing more on learning the categories which are not being correctly defined. This function adds a parameter – gamma -, that *«allows hard-to-classify examples to be penalized more heavily relative to easy-to-classify examples».* The class weights were defined to also provide more weight to the wrongly classified classes. Model 2 provided the following results:

Fig. 8 – Model 2: Summary

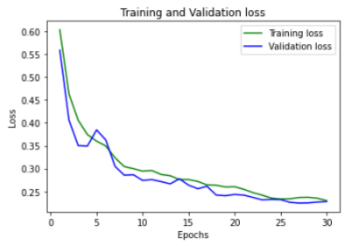
Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_4 (MaxPooling 2D)	(None, 74, 74, 32)	0
conv2d_5 (Conv2D)	(None, 72, 72, 32)	9248
max_pooling2d_5 (MaxPooling 2D)	(None, 36, 36, 32)	0
flatten_2 (Flatten)	(None, 41472)	0
dense_4 (Dense)	(None, 128)	5308544
dense_5 (Dense)	(None, 4)	516

Total params: 5,319,204
Trainable params: 5,319,204
Non-trainable params: 0

Fig. 9 – Model 2: Training and Validation Accuracy

Fig. 10 – Model 2: Training and Validation Loss



The plots are quite similar to the ones from model 1, however, the EarlyStopping function led to a higher number of epochs in this one.

The training accuracy was of 73.1% and the validation accuracy was of 77.7%. This means that we have even more underfitting than in the previous model. From the classification report metrics, we are able to see an increase in the Recall of the Covid class (it learned the minorities better), but the model overall performance decreased.

Fig. 11 – Model 2: Confusion Matrix

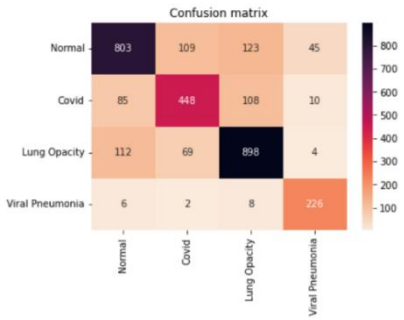


Fig. 12 – Model 2: Classification Report

	precision	recall	f1-score	support
normal	0.80	0.74	0.77	1080
covid	0.71	0.69	0.70	651
lung_opacity	0.79	0.83	0.81	1083
viral_pneumonia	0.79	0.93	0.86	242
accuracy			0.78	3056
macro avg	0.77	0.80	0.78	3056
weighted avg	0.78	0.78	0.78	3056

For the Test set, the accuracy score was of 76.8%.

Model 3

To build our 3rd model, we used both models 1 and 2 and tried to improve them. Although model 2 led to worst accuracy, we figured it would be more interesting to explore it since it led to more balanced results.

For this model, we added more layers to help our model to better learn the images patterns. We also explored some common values (32, 48, 64, 128) for the filter values – following an increasing rate in the neurons' values in the convolution layers and decreasing in the dense layers. In the last pooling layer, we added a stride of (1,1). The reasoning behind this decision was to allow the model to capture more detail in this final layer, before the flatten and dense layers. For this model, we added more dense layers, because sometimes it leads to a better performance of the classifier. The loss function used was the 'SparseCategoricalFocalLoss'. Finally, we also explored the learning rate decay, by using the ReduceLROnPlateau, which reduces the learning rate *«if no improvement is seen for a ''patience' number of epochs».* This helps the network to improve and to learn more complex patterns. This model led to the following outputs:

Fig. 13 – Model 3: Summary

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 148, 148, 32)      896

max_pooling2d_11 (MaxPoolin  (None, 74, 74, 32)       0
g2D)

conv2d_12 (Conv2D)           (None, 72, 72, 48)        13872

max_pooling2d_12 (MaxPoolin  (None, 36, 36, 48)       0
g2D)

conv2d_13 (Conv2D)           (None, 34, 34, 64)        27712

max_pooling2d_13 (MaxPoolin  (None, 17, 17, 64)       0
g2D)

conv2d_14 (Conv2D)           (None, 15, 15, 128)       73856

max_pooling2d_14 (MaxPoolin  (None, 7, 7, 128)        0
g2D)

conv2d_15 (Conv2D)           (None, 5, 5, 128)         147584

max_pooling2d_15 (MaxPoolin  (None, 4, 4, 128)        0
g2D)

flatten_4 (Flatten)          (None, 2048)              0

dense_10 (Dense)             (None, 128)               262272

dense_11 (Dense)             (None, 64)                8256

dense_12 (Dense)             (None, 32)                2080

dense_13 (Dense)             (None, 4)                 132

=================================================================
Total params: 536,660
Trainable params: 536,660
Non-trainable params: 0
_________________________________________________________________
```

Fig. 14 – Model 3: Training and Validation Accuracy

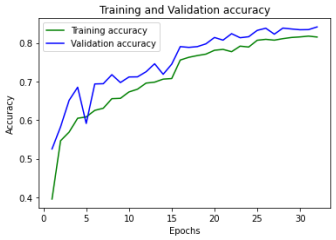
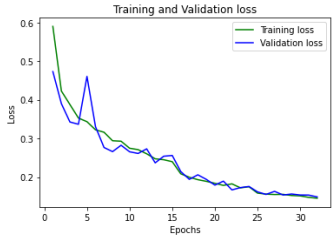


Fig. 15 – Model 3: Training and Validation Loss



We obtained a training accuracy of 81.5%, and a validation accuracy of 84.1%, which was a slight improvement from previous models' accuracies. In addition, we can see a clear decrease in underfitting from model 2. However, we still have a difference of about 3% from the training to the validation accuracy. From the classification report, we can see that the recall increased in all 4 categories.

Fig. 16 – Model 3: Confusion Matrix

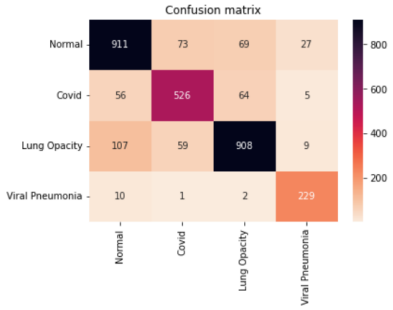


Fig. 17 – Model 3: Classification Report

	precision	recall	f1-score	support
normal	0.84	0.84	0.84	1080
covid	0.80	0.81	0.80	651
lung_opacity	0.87	0.84	0.85	1083
viral_pneumonia	0.85	0.95	0.89	242
accuracy			0.84	3056
macro avg	0.84	0.86	0.85	3056
weighted avg	0.84	0.84	0.84	3056

The Test set accuracy was 83.9%. Since now we already achieved good accuracy and recall scores, we believe it is important to try to solve the underfitting problem our models are facing.

Model 4

In model 4, we added even more layers to our model. We also set padding = 'same' in both convolution layers and pooling layers. We did so to allow our model to control for possible variations in the way the images are presented. We also tried two different loss functions with this model, 'SparseCategoricalFocalLoss' and 'SparseCategoricalCrossentropy', and the latter proved to have better results.

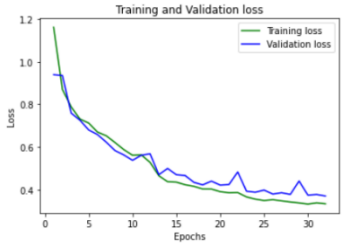
This model led to a training accuracy of 84.4% and validation accuracy of 85.6%.

Fig. 18 – Model 4: Summary

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 148, 148, 32)      896

 max_pooling2d_8 (MaxPooling  (None, 74, 74, 32)       0
 2D)

 conv2d_9 (Conv2D)           (None, 74, 74, 32)        9248

 max_pooling2d_9 (MaxPooling  (None, 37, 37, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 37, 37, 64)        18496

 max_pooling2d_10 (MaxPoolin  (None, 19, 19, 64)       0
 g2D)

 conv2d_11 (Conv2D)          (None, 19, 19, 64)        36928

 max_pooling2d_11 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 conv2d_12 (Conv2D)          (None, 10, 10, 128)       73856

 max_pooling2d_12 (MaxPoolin  (None, 5, 5, 128)        0
 g2D)

 conv2d_13 (Conv2D)          (None, 5, 5, 128)         147584

 max_pooling2d_13 (MaxPoolin  (None, 3, 3, 128)        0
 g2D)

 conv2d_14 (Conv2D)          (None, 3, 3, 128)         147584

 max_pooling2d_14 (MaxPoolin  (None, 2, 2, 128)        0
 g2D)

 conv2d_15 (Conv2D)          (None, 2, 2, 256)         295168

 max_pooling2d_15 (MaxPoolin  (None, 1, 1, 256)        0
 g2D)

 flatten_1 (Flatten)         (None, 256)               0

 dense_3 (Dense)             (None, 128)               32896

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 4)                 260

=================================================================
Total params: 771,172
Trainable params: 771,172
Non-trainable params: 0
_________________________________________________________________
```

Fig. 19 – Model 4: Training and Validation Accuracy

Fig. 20 – Model 4: Training and Validation Loss



Regarding the validation set, we obtained the following results:

Fig. 21 – Model 4: Confusion Matrix

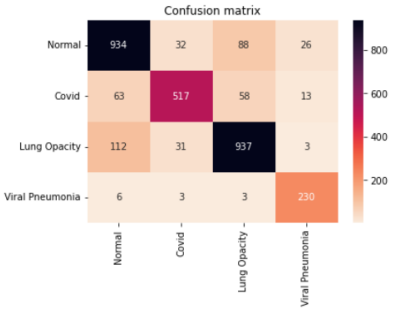


Fig. 22 – Model 4: Classification Report

	precision	recall	f1-score	support
normal	0.84	0.86	0.85	1080
covid	0.89	0.79	0.84	651
lung_opacity	0.86	0.87	0.86	1083
viral_pneumonia	0.85	0.95	0.89	242
accuracy			0.86	3056
macro avg	0.86	0.87	0.86	3056
weighted avg	0.86	0.86	0.86	3056

For the Test set, we obtained an accuracy of 86.6%, and the results were the following:

Fig. 23 – Model 4: Confusion Matrix (Test Set)

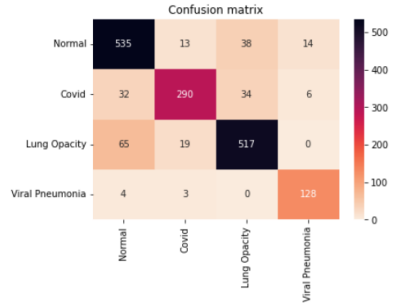


Fig. 24 – Model 4: Classification Report (Test Set)

	precision	recall	f1-score	support
normal	0.84	0.89	0.87	600
covid	0.89	0.80	0.84	362
lung_opacity	0.88	0.86	0.87	601
viral_pneumonia	0.86	0.95	0.90	135
accuracy			0.87	1698
macro avg	0.87	0.88	0.87	1698
weighted avg	0.87	0.87	0.87	1698

Model 4 ended up being our best and final model, since we were able to slightly decrease underfitting and improve the classifier's accuracy. We tried to increase the number of layers in the hope of decreasing underfitting even further, however, this led to worse results in terms of accuracy.

Conclusion

Overall, our final model behaved fairly well in identifying each class – all of them had a recall above 80%. We can see that the class where the model correctly classified more images (in percentual terms) was the Viral Pneumonia one. Even though this was the class with least observations, it did not constitute an obstacle for our classifier.

Although we have very good accuracy scores, since we are dealing with the health sector, it is a very dangerous field to make mistakes. In order to understand the behavior of our model, we tried to do some research regarding the subject. We concluded that, the Covid and Viral Pneumonia x-rays can be confused with each other, since both refer to viruses. However, our model did not find it difficult to distinguish between these two classes. We also discovered that Covid patients can develop lobar pneumonias and ARDS syndrome, both constituting lung opacities. This might explain why 19 images from Covid were classified as Lung Opacity, and why 34 images from Lung Opacity were classified as Covid. We were also able to understand that, in certain cases, Lung Opacity and Normal x-rays can be confused with each other. This explains why between these two classes, the model is wrongly classifying around 100 observations – 38 Normal images predicted as Lung Opacity and 65 Lung Opacity images predicted as Normal.

Finally, our model still had some underfitting. There are some things we could done differently, to try to solve it, namely:

- Increase the input shape – since our model might not be recognizing patterns due to the resolution of the images.
- Extend our CNN even further and possibly explore other values for the filter values both in convolution layers and pooling layers (explore other Pooling approaches, i.e., Average Pooling).
- Explore different models, such as resnet50, VGG16, VGG19, etc.

Although being aware of these possible approaches, due to computational costs and time constraints, we were not able to explore all of them.

Bibliography

Kaggle Dataset Link - https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database

Glob Library to read and retrieve the images - https://docs.python.org/3/library/glob.html; https://stackoverflow.com/questions/51434091/python-globbing-a-directory-of-images; https://www.codegrepper.com/code-examples/python/python+load+all+images+from+folder

Early Stopping – https://keras.io/api/callbacks/early_stopping/

Learning Rate – https://keras.io/api/callbacks/reduce_lr_on_plateau/; https://arxiv.org/abs/1908.01878; https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Weight Class – https://datascience.stackexchange.com/questions/13490/how-to-set-class-weights-for-imbalanced-classes-in-keras

Sparse Categorical Focal Loss – https://focal-loss.readthedocs.io/en/latest/generated/focal_loss.SparseCategoricalFocalLoss.html

Strides – https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Why we need more than one fully connected layer - https://stats.stackexchange.com/questions/212344/why-do-we-have-normally-more-than-one-fully-connected-layers-in-the-late-steps-o

Dense Layers – https://stats.stackexchange.com/questions/338255/what-is-effect-of-increasing-number-of-hidden-layers-in-a-feed-forward-nn

Underfitting – https://allcloud.io/blog/how-to-solve-underfitting-and-overfitting-data-models/#:~:text=Increasing%20the%20model%20complexity&text=Using%20a%20more%20complex%20model,very%20often%20help%20solve%20underfitting