Project: COVID-19 Assisted Diagnosis

Carlo Heemeryck[1] Willem Van Nieuwenhuyse[1] Seppe Vanrietvelde[2] Luca Visser[2]

[1] Bachelor of Science in de ingenieurswetenschappen - werktuigkunde-elektrotechniek
[2] Master of Science in Statistical Data Analysis - Computational Statistics

I. Introduction

The outbreak of the COVID-19 pandemic in late 2019 has placed unprecedented strain on healthcare systems worldwide, highlighting the urgent need for rapid and accurate diagnostic tools to support clinical decision–making. Chest X-ray imaging has emerged as a widely available modality for the preliminary screening of COVID-19 pneumonia, yet visual assessment alone can be time-consuming and prone to inter-observer variability [**?**], [**?**]. In this work, a deep learning–based framework for assisted diagnosis of COVID-19 from chest X-ray images is presented, leveraging both custom convolutional neural networks and transfer learning approaches.

Section **??** covers exploring the publicly available dataset, detailing the preprocessing pipeline (downsampling, normalization, and light augmentation) and presenting initial sample images (Fig. **??**). In Section **??**, a baseline CNN model is build and evaluated, reporting its architecture (Table **??**), learning curves (Fig. **??**), and test-set performance including a confusion matrix (Fig. **??**). Section **??** introduces transfer learning with a pre-trained ResNet50V2 backbone, describing model adaptations, hyperparameter tuning, and comparative results. Section **??** demonstrates model interpretability using Grad-CAM heatmaps, and finally Section **??** summarizes the conclusions and outlines potential future work.

II. Task 1: Data Exploration, Pre-Processing and Augmentation

A. Loading the data

The grayscale chest X-ray images are loaded at their original resolution of 299 × 299 pixels. The labels, either COVID or NORMAL, are taken from the folder names containing the images and assumed to be accurate. From the outset, the dataset was pre-divided into training (1,600 images), validation (400 images), and test (200 images) sets. This original distribution is preserved for the exercise, with an additional combined training and validation set comprising 2,000 images.

B. Data exploration

Exploration consists of checking the shape of the data, evaluating the distribution of the two classes, plotting a few examples and checking pixel and global average and standard deviation. This is replicated for training, validation and test datasets and serves as an initial look at the data and to possibly detect problems at an early stage in the project. All the images in the training dataset are of the same size and there are an equal amount in both classes, which is good. A few labeled examples are given in Fig. **??**.

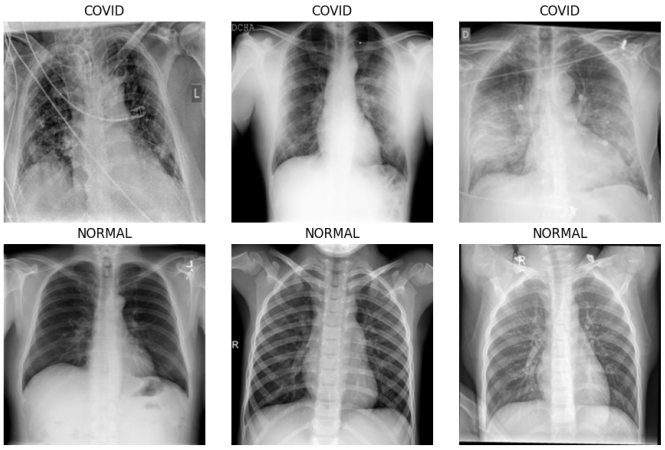


Fig. 1: A few images from the training dataset with accompanying labels.

There seem to be no clear differences between the NORMAL and COVID images. However, the limitation to grayscale images, low contrasts, different zoom levels and slight variations in angles could be problems for classification. Also if there are, to us unrecognisable, artifacts in mostly one category, generalization to new images could be problematic. Over all images in the training dataset the average value and the standard deviation at every pixel location can be visualized to show a sort of average image and a heatmap for variation. These visualizations, together with the average and standard deviation of all pixels, are given in Fig. **??**.

Replicating all this for the validation and test datasets give very comparable results, indicating that the images were uniformly divided over the different sets. The exploration of the validation and test sets can be consulted in the notebook.

C. Pre-processing

Next, some pre-processing steps are implemented and evaluated, namely down sampling to 128 x 128 pixels using bilinear interpolation and a few normalization strategies. The result of down sampling, given in Fig. **??**, is still recognisable to the human eye. As such, it is assumed that the model still

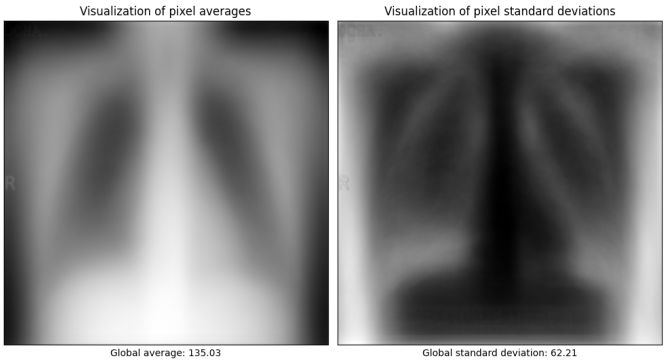


Fig. 2: A visualization of the average image (left) and a variation heatmap (right) with accompanying global statistics from the training dataset.

gets enough information from this. Implementing this down sampling strategy should reduce training time while not losing too much performance.

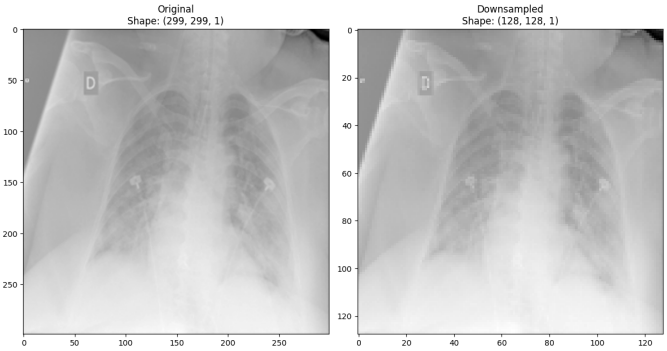


Fig. 3: An example of a downsampled image using bilinear interpolation (299x299 -¿ 128x128).

Normalization through the use of a fixed value (dividing by 255), sample statistics (subtracting the overall mean and dividing by the overall standard deviation of the current sample (training, validation or test)) and statistics of the training dataset (subtracting the overall mean and dividing by the overall standard deviation of the training dataset) were tried out. All of these strategies resulted in visually the same image but with pixel values with a far lower magnitude. However, normalizing using training set-wide image statistics (mean and std) and applying the same normalization across all images is the most correct method as it causes no data leakage. Normalization based on the training dataset will thus be used in the project.

D. Augmentation

Finally, based on the variety of the plotted images a few augmentation strategies are implemented, namely randomly altering the brightness and the contrast by a factor between 0.6 and 1.4 and randomly rotating the images (anti)clockwise up to 30°. Examples of these augmentation are given in Fig. **??**.

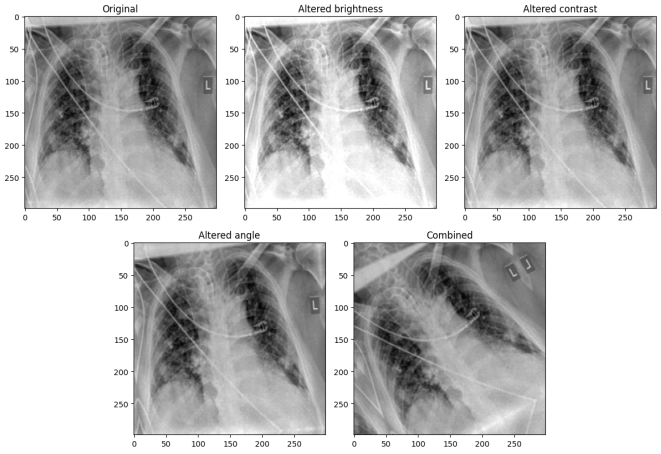


Fig. 4: Examples of different transformations as well as the combination of them together.

Augmentation should only result in images that could occur in test/unseen data. Based on the data we have, intense augmentation doesn't seem al that necessary. It will thus only be considered if necessary.

E. Pipeline

A pipeline consisting of loading the images correctly, with optional downsampling, followed by optional normalization using the training dataset is implemented for later use. Data augmentations can be used as layers in the models if necessary.

III. Task 2: Building the Baseline Model

This section describes the construction, training, and evaluation of an initial convolutional neural network (CNN) model for COVID-19 detection from chest X-ray images. Starting with outlining the model architecture, then proceeding to tuning hyperparameters, and concluding with a performance analysis on the validation and test sets.

A. Initial Model Architecture

A custom CNN with the following components is implemented:

- **Input layer**: $128 \times 128 \times 1$ grayscale image
- **Convolutional blocks**: three blocks with increasing filter sizes $\{16, 32, 64\}$, each block consists of a 3×3 convolution, ReLU activation, max-pooling, and dropout (rate 0.1)
- **Fully connected layer**: 512 units with ReLU activation and dropout (rate 0.1)
- **Output layer**: single unit with sigmoid activation for binary classification

A summary of the full architecture is shown in Table **??**.

TABLE I: Initial Baseline CNN Architecture Overview.

Layer (type)	Output Shape	# Params
Conv2D (3×3, 16 filters)	(None, 128, 128, 16)	160
MaxPooling2D (2×2)	(None, 64, 64, 16)	0
Dropout (rate = 0.1)	(None, 64, 64, 16)	0
Conv2D_1 (3×3, 32 filters)	(None, 64, 64, 32)	4,640
MaxPooling2D_1 (2×2)	(None, 32, 32, 32)	0
Conv2D_2 (3×3, 64 filters)	(None, 32, 32, 64)	18,496
MaxPooling2D_2 (2×2)	(None, 16, 16, 64)	0
Dropout_1 (rate = 0.1)	(None, 16, 16, 64)	0
Flatten	(None, 16·16·64 = 16384)	0
Dense (512 units)	(None, 512)	8,389,120
Dense_1 (1 unit, sigmoid)	(None, 1)	513
Total parameters		**8,412,929**
Trainable parameters		**8,412,929**
Non-trainable parameters		**0**

B. Training Procedure

The model is initially trained on batch sizes of 32 using a learning rate of 10^{-3} using the Adam optimizer on a binary cross-entropy loss function and accuracy as the metric of choice.

The learning curves, spanning 30 epochs, are depicted in Fig. **??**.

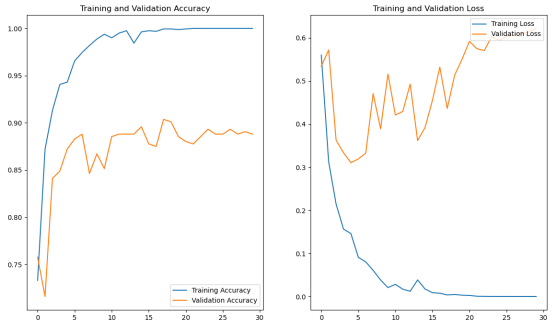


Fig. 5: Training and validation accuracy and loss for the baseline CNN over 30 epochs.

C. Hyperparameter Tuning

A grid search (with early stopping) was performed over the following hyperparameters:

- Dropout rate: {0.1, 0.2}
- Initial convolutional filters: {8, 16}
- Learning rate: {1e-3, 5e-4}
- Dense units: {256, 512}

The best configuration achieved a validation accuracy of **0.8932** with a dropout rate of **0.2**, **8** filters, a learning rate of $\mathbf{10^{-3}}$, and **512** dense units. All results are shown in Table **??**.

D. Final Baseline Model

Using the best hyperparameters, the model was retrained on the training and validation sets together (2,000 images) for 8 epochs. Evaluation on the held-out test set (200 images) yielded:

TABLE II: Hyperparameter Tuning Results (sorted by validation accuracy).

Dropout	Filters	LR	Dense Units	Val Acc	Epoch	Val Loss
0.2	**8**	**1e-3**	**512**	**0.8932**	**9**	**0.2984**
0.2	16	1e-3	256	0.8906	7	0.2998
0.1	16	5e-4	512	0.8828	6	0.2970
0.1	8	1e-3	512	0.8828	12	0.3483
0.1	8	1e-3	256	0.8776	8	0.3139
0.1	16	1e-3	512	0.8724	5	0.3096
0.2	16	5e-4	256	0.8698	9	0.3422
0.2	8	5e-4	512	0.8698	10	0.2969
0.2	16	5e-4	512	0.8672	4	0.3052
0.1	8	5e-4	512	0.8646	12	0.3483
0.2	16	1e-3	512	0.8646	6	0.3223
0.1	16	1e-3	256	0.8620	4	0.3276
0.2	8	5e-4	256	0.8620	8	0.3166
0.1	8	5e-4	256	0.8594	8	0.3343
0.2	8	1e-3	256	0.8464	5	0.3387
0.1	16	5e-4	256	0.8385	5	0.3608

- Test accuracy: 0.8300
- Test loss: 0.3835

The confusion matrix is shown in Fig. **??**.

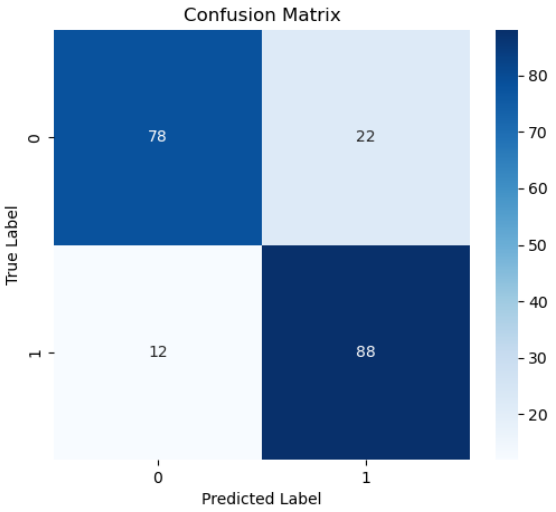


Fig. 6: Confusion matrix of the final baseline model on the test set (1: NORMAL, 0: COVID).

E. Discussion

The baseline model achieved 83% accuracy, demonstrating strong discriminative power between COVID-19 and normal chest X-ray images. The following is observed from the confusion matrix (Fig. **??**):

- True negatives (NORMAL correctly classified): 88
- False positives (NORMAL misclassified as COVID): 12
- False negatives (COVID misclassified as NORMAL): 22
- True positives (COVID correctly classified): 78

This yields a sensitivity (recall for COVID) of $78/(78+22) = 0.78$ and a specificity of $88/(88+12) = 0.88$. The slightly higher sensitivity indicates that the model is more likely to correctly detect COVID cases, at the cost of some false alarms.

In our final training run—conducted without a separate validation split—we noted that training loss continued to decline predictably with each epoch. However, without a validation curve to signal overfitting, determining the optimal early-stopping point became speculative. In future work, reintroducing a validation set or implementing cross-validation will be essential to choose the right number of epochs.

The training curves, shown in Fig. **??**, suggest rapid convergence of training accuracy to nearly 100% within the first 10 epochs, while validation accuracy plateaus around 88%, suggesting that there is still some overfitting, even after controlling via dropout. Overall, the baseline CNN provides a solid starting point.

IV. Task 3: Transfer Learning

In this task transfer learning by using a pre-trained ResNet50V2 model trained on Imagenet is applied. Transfer learning is a powerful machine learning technique for leveraging knowledge learned from one task and applying it to a new task. This is especially useful when you have limited training data.

The typical workflow for transfer learning is as follows:

1) Start with a pretrained model
2) Freeze the layers of the pretrained model
3) Add new trainable layers to the pretrained model
4) Train on the dataset
5) Fine-tuning (optional)

The layers of the pretrained model are frozen to prevent elementary information learned from the initial task from being overwritten during the new training process. New layers are added to the pretrained model to adapt it to the new task, which need to be trained on new data. Optionally, the last layers of the original model can be unfrozen and retrained to fine-tune the model even more.

A. Setup of the base model

The base model (ResNet50V2) is instantiated with the "imagenet" pre-trained weights, not including the top layers. All the layers of the base model are frozen so their weights are not updated during the training process. Then, new fully connected layers are added to the model with dropout in between (Table **??**). The architecture of the final layers added to the pre-trained model are exactly the same as the fully connected layers of the baseline model.

TABLE III: Initial Transfer Model Architecture

Layer (type)	Output Shape	# Params
augmentation	(None, 128, 128, 3)	0
ResNet50V2 (pre-trained)	(None, 4, 4, 2048)	23,564,800
Dropout (rate = 0.1)	(None, 4, 4, 2048)	0
GlobalAveragePooling2D	(None, 2048)	0
Dense (256 units)	(None, 256)	524,544
Dropout (rate = 0.1)	(None, 256)	0
Dense (256 units)	(None, 256)	65,792
Dropout (rate = 0.1)	(None, 256)	0
Dense_1 (1 unit, sigmoid)	(None, 1)	257
Total parameters		**24,155,393**
Trainable parameters		**590,593**
Non-trainable parameters		**23,564,800**

B. Training procedure

The model is initially trained using the best hyperparameters found while tuning the baseline model in the previous section, again using the Adam optimizer on a binary cross-entropy loss function and accuracy as the metric of choice.

The learning curves, spanning 30 epochs, are depicted in Fig. **??**.

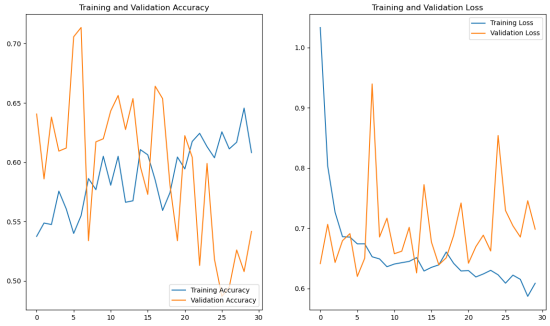


Fig. 7: Training and validation accuracy and loss for the initial transfer learning model over 30 epochs.

The model trains without errors and the loss stabilizes somewhat but the results are clearly worse.

C. Hyperparameter tuning

A grid search (with early stopping) was performed over the following hyperparameters:

- Dropout rate: {0.1, 0.2, 0.3}
- Learning rate: {1e-3, 5e-4}
- Batch size: {32, 64}

The best configuration achieved a validation accuracy of **1.0000** with a dropout rate of **0.1**, a learning rate of $\mathbf{1 \times 10^{-3}}$, and a batch size of **64**.

D. Final transfer learning model

Using the best hyperparameters, the model was retrained on the training and validation sets together (2000 images) for 2 epochs.

the learning curves, spanning 2 epochs, are depicted in Fig. **??**.

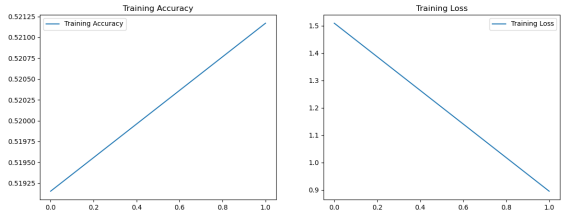


Fig. 8: Training accuracy and loss for the final transfer learning model over 2 epochs.

E. Fine-tuning

In this step all of the base model layers are unfrozen. The whole model is completely retrained using the training and validation set together.

Evaluation on the held-out test set (200 images) yielded:

- Accuracy: 83.5%
- Loss: 0.6901

The model clearly performs better than the pre-trained model. This could be expected because the model is now completely trained on our own data.

The confusion matrix is shown in Fig. **??**.

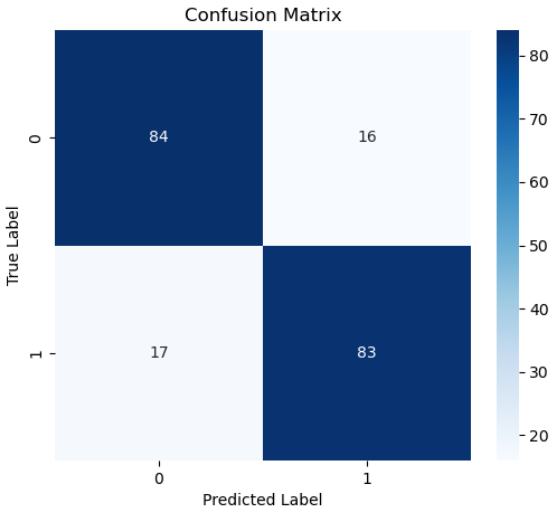


Fig. 9: Confusion matrix for the fine-tuned transfer learning model (1: NORMAL, 0: COVID).

Finally a few images are plotted from the test dataset in Fig. **??**. First without pre-processing, then with their evaluations after pre-processing.

F. Discussion

The following is observed from the confusion matrix (Fig. **??**):

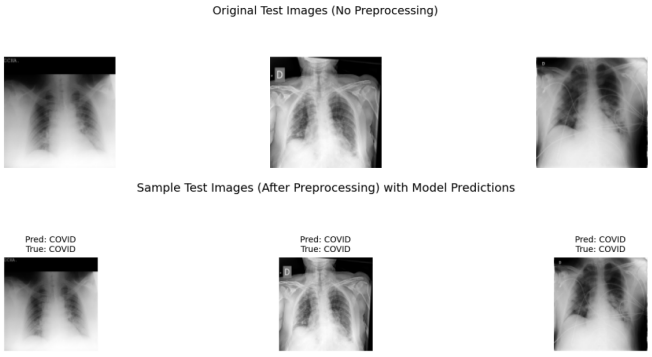


Fig. 10: Test Samples with and without pre-processing and there labels.

- True negatives (NORMAL correctly classified): 84
- False positives (NORMAL misclassified as COVID): 17
- False negatives (COVID misclassified as NORMAL): 16
- True positives (COVID correctly classified): 83

This yields a sensitivity (recall for COVID) of $83/(83+17) = 0.83$ and a specificity of $84/(84+16) = 0.84$. Overall, the model performs reasonably well but could still benefit from reducing false positives and false negatives.

The results are very similar as the baseline model. But being more complex makes it less preferred.

From this experience, it can be concluded that transfer learning has both advantages and disadvantages. On the positive side, training was faster given the same amount of data. On the negative side, the base model and its pre-trained weights may not have been well-suited for the task at hand, making fine-tuning a delicate and challenging process.

To improve the effectiveness of transfer learning in this context, several modifications could be considered. Notably, the base model was pretrained on RGB images of various objects, which differs significantly from the grayscale medical images used in our task. Using a pretrained model that was trained on data more similar to our target domain could yield better results and enhance the overall performance of the transfer learning approach.

V. Task 4: Explainability through Grad-CAM

In this section, we analyse how our COVID-19 classifier makes decisions. We will do so by introducing a technique called Gradient-weighted Class Activation Mapping (which, from now on, we will refer to as Grad-CAM), which allows us to visually analyse what parts of the input images the model focusses on when making decisions about the presence of COVID-19.

We will generate heatmap overlays on test images using the Grad-CAM technique. This way, we will be able to see which regions of the lungs are of more importance to the classifier than others. We hope to see highlights in COVID-relevant regions, as this would mean our model accually learned to detect COVID-19 as opposed to relying on cheats or

artifacts. Furthermore, we hope to see a difference in activation patterns when comparing normal and affected lungs during our analysis.

A. Grad-CAM implementation

To realise a Grad-CAM implementation, we mostly follow the Keras tutorial, adjusted for binary classification.

We use the pre-trained model from Task 2 (the baseline model), as it has delivered us the best test performances. The model's architecture overview can be found in Table **??**.

Removing the final layer's activation (sigmoid in our case) is the next crucial step in setting up our model for Grad-CAM. This avoids suppressing the gradient into a small range, which would cause for meaningless heatmaps.

Next, we select a test sample from our test batch and pre-process it by normalising and adding a dimension as to transform it into a 'batch' of size one. Using this test image, we evaluate our model and obtain the last convolutional layer's output. This layer's output is specifically neccessary to compute the gradients of the predicted class, because it preceeds the final layers' flattening, thus preventing the loss of spacial information.

The next step is to compute the gradients of the predicted class with respect to the feature map of the last convolutional layer's output we just obtained. This is where we adjust the Keras tutorial to fit binary classification by setting the prediction index to zero.

Now we follow the prewritten code in the Keras tutorial to complete the following steps: obtaining the heatmap by averaging the gradients along the spacial dimension and using these as weights for channel importance, normalising the heatmap between 0 and 1 and mapping it onto the original sample space, and finally overlaying the heatmap onto the original image by adding to its values rescaled by 0.4.

The heatmap we obtain by applying our Grad-CAM implementation to our test image is shown in Fig. **??**. We conclude that it works correctly.

B. Explaining the decision with Grad-CAM

Firstly, when we analyse the test image's Grad-CAM, we see that the entirety of the lungs are highlighted. This is both expected, as we want our model to detect COVID-19 indications within the lungs, and also a possible concern, as there is no specific region within the lungs that seems to be prodominent in the decision–making of our model. Yet another reason for concern are the highlighted areas around the edges of the scan. Our classifier seems to focus a lot on areas which are not a part of the X-ray, areas that do not provide any additional information as to whether COVID-19 is present or not.

To explain the way our classifier makes decisions, we need to further divide our test dataset. We devide our dataset into four distinct groups:

- Normal lungs (True Negative)
- Affected lungs (True Positive)

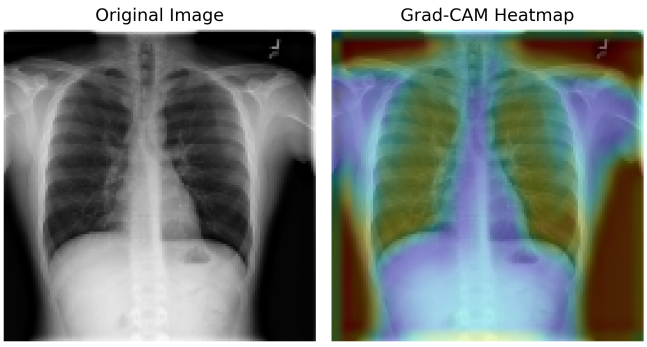
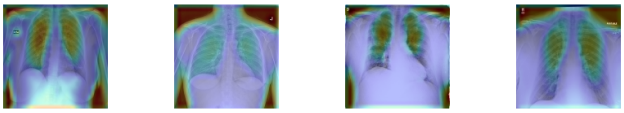


Fig. 11: Grad-CAM of the test image.

- Normal lungs that were misclassified by our model (False Positive)
- Affected lungs that were misclassified by our model (False Negative)



(a) True Negative (b) True Positive (c) False Positive (d) False Negative

Fig. 12: Grad-CAM visualizations per classification group.

We've taken one image from each of the four groups, as shown in Fig. **??**. These images mostly tell us that our model always looks at the edges of the scans, which isn't desired. Setting aside the highlighted edges, we can also see that the lungs are highlighted.

Let's first look at the correctly classified images. In the normal lungs (**??**), the model doesn't seem to take interest in specific parts of the lungs, it just overlooks them generally. While in the affected lungs (**??**), the highlighted regions seem to be more specific. We chose this example for affected lungs, because it seems that the lungs are actually not fully highlighted. This could mean our model made a 'lucky guess' based off of artifacts (like the 'L'-shaped indicator or the edges of the scan). This is a problem because it shows us that the classifier (partially) makes decisions based off of information which has nothing to do with the state of the lungs, thus not providing a reliable classification. It shows us that the model isn't perfect and needs more training (data).

Now, let's look at the misclassified images. In the case where normal lungs were misclassified (**??**), we can see the classifier focussing again on specific parts of the lungs, but being missled into thinking the lungs contain COVID-related anomalies. When we look at the affected lungs which our classifier wrongly assumed were normal (**??**), we can see that

the highlighted regions are much more general. Either the X-ray doesn't show clear enough signs of COVID-19, or our model is not yet optimally trained.

C. Discussion

Because we work with a binary classifier, our model outputs a single sigmoid logit rather than a full vector of class scores, we removed the final activation so we'd get an unsquashed logit and then treated that scalar as the "class score" when computing gradients. Otherwise, the core Grad-CAM steps remain unchanged.

Furthermore, by having only two class labels, the highlights on our Grad-CAM simply show which regions push the model toward "positive" versus "negative." There's no need to compare multiple classes, so the heatmap directly reflects the single decision boundary, which makes interpretation simpler but less nuanced than in a multiclass setting.

In well-trained cases, the heatmaps tend to light up lung regions where COVID-related opacities appear (e.g., ground-glass patterns or consolidations). In our case we rarely see such specific regions highlighted (perhaps due to limited data and limited training time). We are also seeing activations outside the lungs, which tells us that the model might be overfitting or might just be focusing on the wrong features.

In this way, the Grad-CAM visualizations did help us identify potential problems in our model. By inspecting where the model focuses, we can spot biases. For example, if Grad-CAM repeatedly highlights text labels, borders, or equipment artifacts instead of lung tissue, that indicates the model is learning shortcuts tied to dataset artifacts rather than true pathology. The Grad-CAM vizualisations were also especially helpful for misclassifications: for false positives it often lights up benign findings (like old scars), and for false negatives it shows when the model simply missed subtle opacities. Seeing these patterns guides us to understand exactly why a sample was mistaken.

If we could collect additional data and retrain the model, we would collect more examples of under-represented disease patterns (e.g., peripheral opacities), remove or crop out distracting artifacts (like text markers), and balance data from different hospitals. This targeted data curation and augmentation would help the model learn to focus properly on medically relevant lung features.

VI. Conclusions

Our preprocessing pipeline's success in preserving diagnostic detail while standardizing inputs means the model is learning from consistent, clinically relevant features rather than spurious artifacts. By downsampling images to 128×128, we struck a balance between computational efficiency and preserving the lung patterns critical for detecting COVID-19 infiltrates. The uniform distribution across dataset splits—and the fact that light augmentations did not introduce unrealistic distortions—suggests that our model will be resilient to variations in real-world chest X-ray quality (different machines, exposure settings, and patient positioning), reducing the risk that it will latch onto dataset-specific quirks rather than true pathology.

Moving to the baseline CNN, reaching high sensitivity with somewhat lower specificity highlights an important clinical trade-off: the model is tuned to minimize missed COVID-19 cases (false negatives), which is crucial in a screening context, even at the cost of some false positives. This means in practice, our tool could flag more scans for follow-up rather than risk overlooking an infected patient, aligning with a conservative screening philosophy. The rapid convergence and plateauing of validation performance indicate that our architecture is powerful enough to capture discriminative patterns but also that future gains will likely come not from making the network deeper, but from richer data (additional cases, varied sources) or from integrating more advanced techniques like transfer learning and interpretability methods to further refine what the model "looks at" in each X-ray.

VII. Author contributions and collaboration

In our group-based collaboration, each member took primary responsibility for one of the four major tasks—Seppe led Task 1 (Data Exploration and Pre-Processing), Luca developed Task 2 (Baseline Model Architecture), Willem implemented Task 3 (Transfer Learning), and Carlo focused on Task 4 (Grad-CAM Explainability)—while all four contributed together to framing the introduction, discussion, and conclusion. Throughout the project, we regularly cross-reviewed each other's code, methodologies, and report drafts to ensure consistency, correctness, and clarity, stepping in to proofread and validate results beyond our individual assignments.

VIII. Use of Generative AI

In this project, Practical 3 served as the foundational template for Tasks 1 & 2—providing the baseline model architecture, image-generation pipeline, and core preprocessing routines—while ChatGPT and GitHub Copilot were primarily leveraged to tidy up code, help identify bugs, and offer small implementation suggestions. Neither tool was purposefully relied upon to devise fundamentally new methodologies or novel, overly complex model designs beyond the scope of the course.

For the written report, ChatGPT was also used sparingly to automate repetitive formatting tasks such as inserting tables and to perform spell-checking and grammar refinement suggestions, ensuring consistency and polish without supplanting the original content.

A. Figures and Tables

a) Positioning Figures and Tables: All figures and tables are inserted using the placement specifier `[htbp]`; each graphic is scaled to the column width via `\includegraphics[width=\linewidth]{...}`, centered in its float, and accompanied by a concise caption via `\caption{...}`. We reference every float in the text (e.g., "Fig.X" or "TableY") and rely on the class's default top-of-column placement to maintain formatting consistency.

REFERENCES

[1] Wang, L., Lin, Z. Q., & Wong, A. (2020). Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. Scientific reports, 10(1), 19549.

[2] Apostolopoulos, I. D., & Mpesiana, T. A. (2020). Covid-19: automatic detection from x-ray images utilizing transfer learning with convolutional neural networks. Physical and engineering sciences in medicine, 43, 635-640.