Analysis of Covid-19 chest X-rays

Data preprocessing and modelling report

ALEXANDRA RANCIC, PHILIPP TRINH, PREETHA BALAKRISHNAN,
PAUL POURMOUSSAVI

1 INTRODUCTION

The goal of this project is to develop a deep-learning model for the detection of COVID-19 based on sample X-ray images of the chest region. As the first step towards this goal, we previously explored the data at hand and observed the following:

1) The X-ray images were 299*299 pixels in size
2) The corresponding masks were 256*256 pixels in size
3) The number of images and masks were equal
4) The number of images per category were not equal indicating that they must be weighted accordingly before training the model
5) The overlay between the images and corresponding masks worked well indicating that one could use it to mainly focus on the lung area of the image

In the current report, we will discuss in detail about image pre-processing that is imperative to develop a robust model, model development (deep and machine learning models) and model optimization.

2 METHODLOGY

Sample X-ray image data required to train the model were obtained from Kaggle, a large machine learning and data science online platform that helps people to build their skills with various data-related challenges. It contains '.png' images and corresponding masks of volunteers/patients grouped into four categories: normal (i.e. images from healthy volunteers), viral pneumonia, lung opacity, and COVID-19. Raw images and masks were studied and further processed step-by-step to make it more compatible for machine and deep learning using python.

3 RESULTS

We divided the observations of this report into two parts. The first part mainly focuses on the pre-processing of the X-ray images and the second part delved deep into the various models we tested.

3.1 Image pre-processing

The used dataset, containing X-rays (named images) and masks, is not balanced, as shown in Figure 3.1.

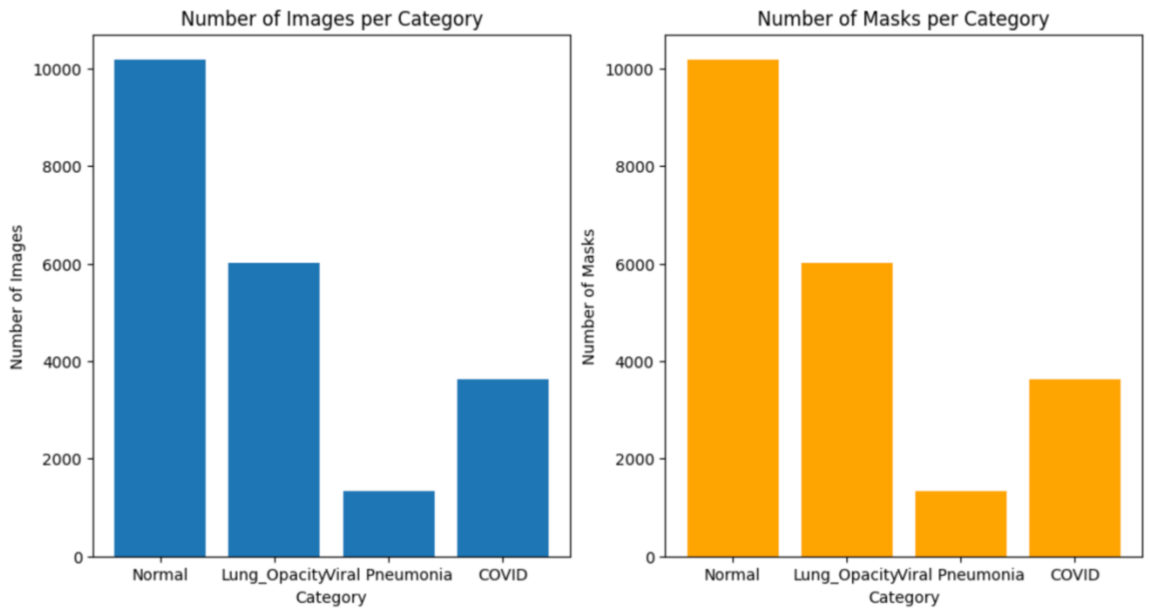


Figure 3.1. Visualization of number of images (left, blue) and masks (right, orange) per category.

This is important since most of the Machine Learning algorithms are based on the assumption that the dataset is balanced. Balanced dataset is the dataset with data equally distributed among all of its classes. Training a model on an imbalanced dataset is surely possible. However, the learning becomes biased towards the majority classes.

The imbalanced dataset containing images could be handled with Sample Weighting in Loss Function. The method requires to weight the loss computed for different samples differently, depending whether they belong to the majority or the minority classes. In the simple words, more priority is given to the category having less samples.

Other possible techniques to handle imbalanced dataset are oversampling and undersampling, and generating synthetic samples, encoding the labels, and splitting the data.

In the first part of this analysis, we were focused on the statistical weighting of our dataset. Statistical or class weighting is straightforward to implements. It does not alter the number of samples, unlike oversampling techniques such as Synthetic Minority Oversampling Technique (SMOTE) or undersampling that can artificially and randomly inflate or reduce dataset size. Statistical weighting reduces the risk of overfitting, since it does not create duplicate samples or synthetic data. This cannot be said for oversampling with SMOTE. It is important to underline that statistical weighting is compatible with most algorithms, classical such as Linear Regression, Random Forest, or Boosting, but also with Convolutional Neural Networks (CNN).

The class weights were calculated on the dataset previously split with the test size of 20%. From the total of 21165 images, 16932 images were in the training set, and 4233 in the testing set. Before splitting the model, first we checked whether all the images are grayscale. Since this was not the case with all images in the category ***Viral Pneumonia***, the code for converting them into grayscale was run. Then, images were converted into np.arrays (X and y). By flattening the images, 2D images were transformed into 1D arrays.
Figure 1.2. visualize the connection class distribution in training set and computed class weights.

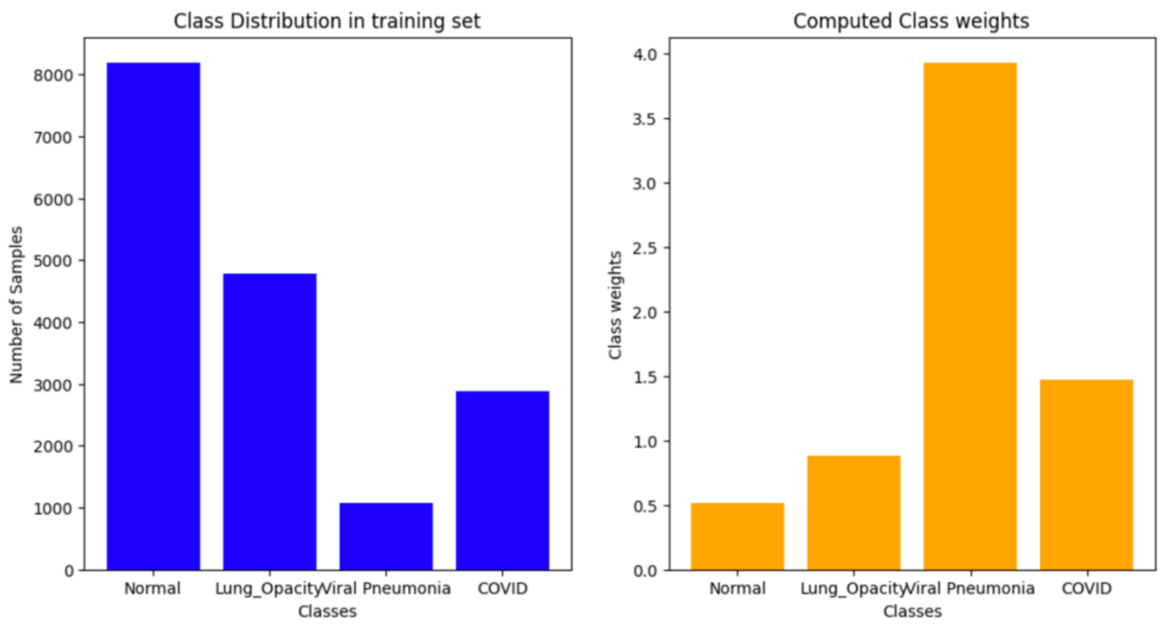


Figure 3.2. The distribution of images in training set (left, blue) and computed class weights (right, orange)

From Figure 1.2., we can conclude that the most of priority is given to the category named ***Viral Pneumonia***, since the number of images in this category is the lowest. On the other side, the least priority is given to the category ***Normal***, since its number of images is the highest.

Furthermore, before starting with Machine Learning, the standard scaler was imported. Standardizing the images ensures uniform scale, or in the other words, that each pixel value has a similar range. It helps also mitigating the effect of different lighting conditions and

shadows, making the model more robust to variation in image capture conditions. This may be particularly important for X-ray datasets. Standard scaler also reduces the influence of outliers (extreme pixels values), which may distort the learning process.
In the first part of classical modelling, we were focused on the whole images (X-rays), and for the second part of classical modelling we used Region of Interest (lungs) and filters.

3.1.1 Selecting the lung area in images

Corona virus primarily resides in the lungs and leads to complications in this region. For this reason, we decided to use this area as our region of interest. To obtain this region of interest, we first resized the raw X-ray images to 256*256 pixels in size. Using the corresponding masks of the same size, we extracted just the lung area of the X-ray of each image. This procedure was followed for every image from each category (Figure 3.3., shows a sample image from each category). This region of interest in grayscale was used in the subsequent pre-processing steps.

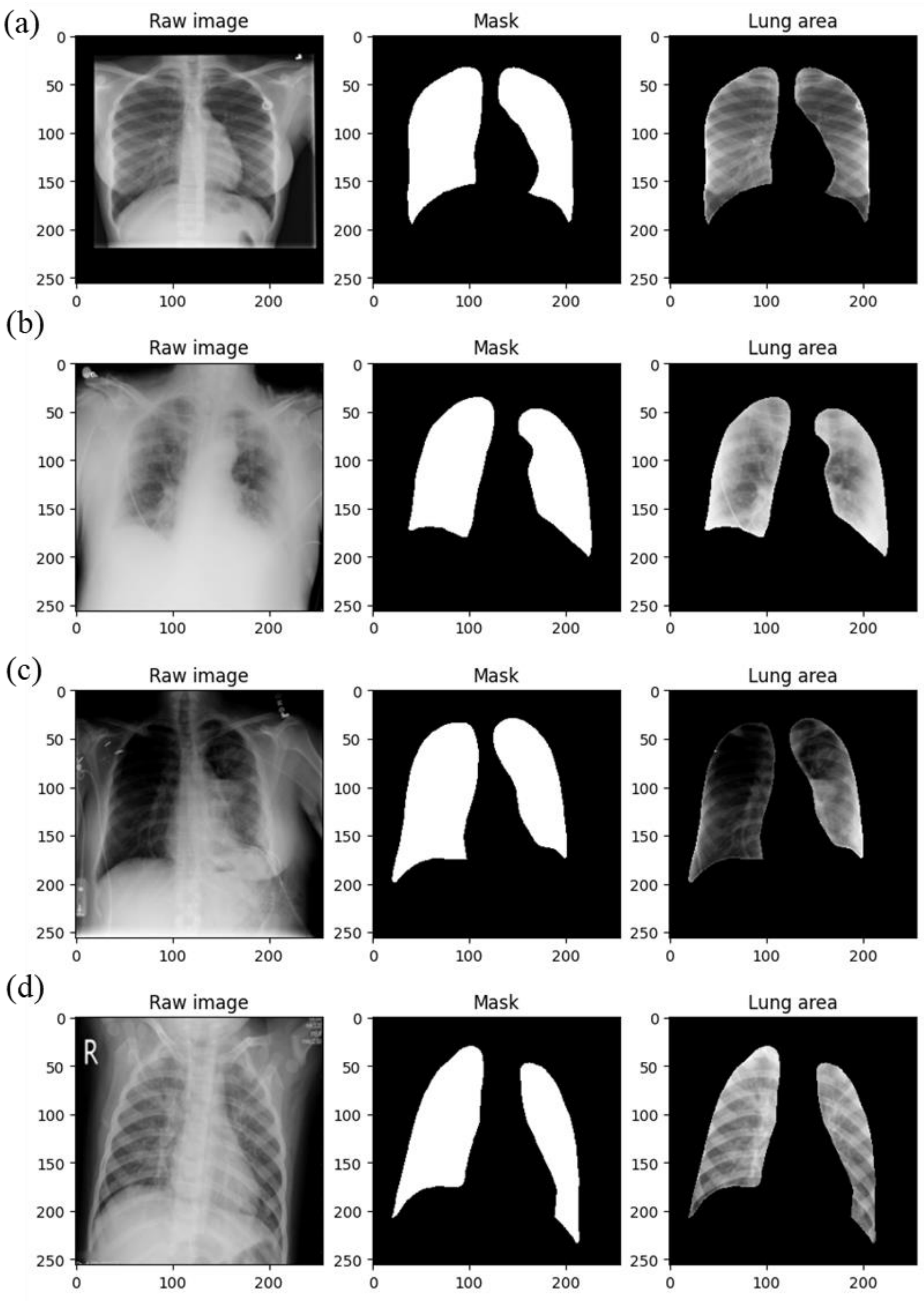


Figure 3.3. Selecting the lung region of the chest X-ray (a) Representative X-ray of a normal volunteer (b) X-ray image of a COVID-19 positive patient (c) X-ray image of a patient with an opaque lung (d) X-ray image of a patient with pneumonia. The scale signifies the width and height of the image in pixels.

3.1.2 Image processing

Image processing is one of the crucial steps before using the data for training any machine learning (ML) or deep learning (DL) model. Raw image data has ample noise that could unnecessarily delay the ML or DL learning time that in future could be costly for hospitals and diagnostic centers. Noise could also decrease the precision and accuracy of the model in question. Therefore, we tried various filtering techniques that could reduce the noise and improve the disease detection capacity of the model. To understand how each filter visually altered the image, we combined every filtering step with a Canny edge detection step.

3.1.2.1 The median blur filter

The median blur filter finds the median value in the neighborhoods of each pixel. It runs through each pixel and replaces it with the median value. We tested this filter using three different kernel sizes – (3, 3), (5, 5) and (7, 7). As shown in Figure 3.4. (a), the kernel size (5, 5) and (7, 7) blurred the image considerably and a subsequent Canny edge detection (see section 3.1.2.5 for details about Canny edge detection) of these images showed that the edges of the ribcage were not entirely detected, and it sort of introduced shapes in the images that were not represented in the original image (Figure 3.4. (b)). Therefore, a kernel size of (3, 3) was the best as it retained the structures of the original image and reduced the noise in the image.

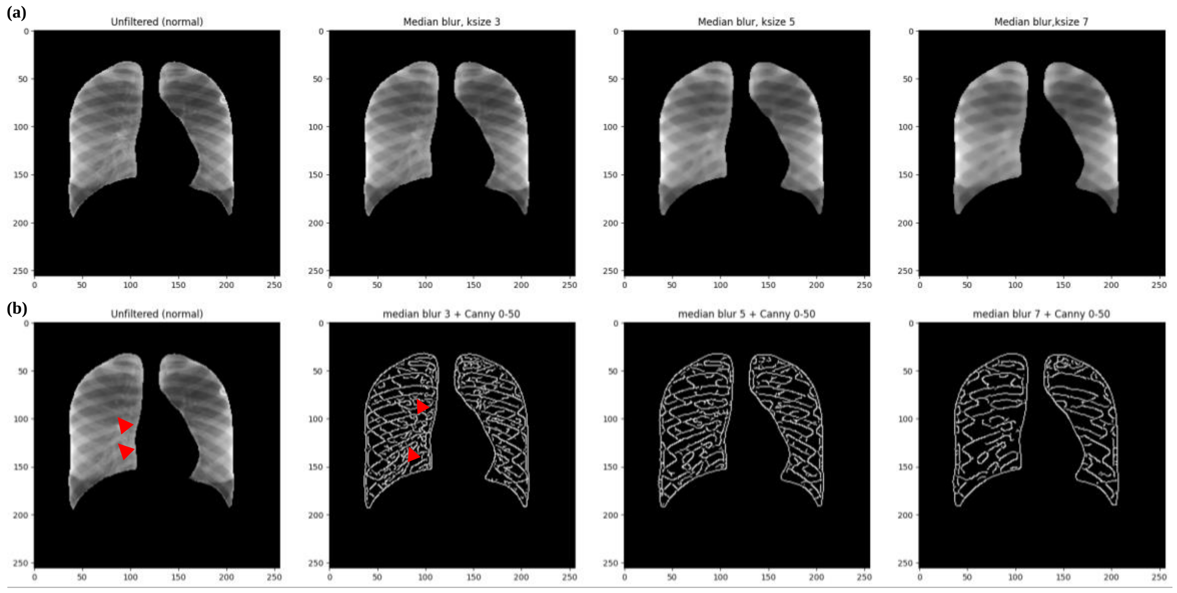


Figure 3.4. Median blur and Canny edge detection (a) Representative lung area of a normal volunteer and median blur filtering using a kernel size (or ksize) of (3, 3), (5, 5) or (7, 7) (b) Representative lung area of a normal volunteer and median blur filtering using the different kernels followed by a Canny edge detection (min = 0, max = 50). The scale signifies the width and height of the image in pixels. The arrow in red shows the thin fibre like bronchus edges detected with a Canny edge detector.

The maximum threshold value used for Canny edge detection, for median blur is 50. With this value, even thin, tree like bronchus could be detected strongly (Figure 2b). This may interfere with model prediction (additional noise) in future and hence, a maximum value greater than 50 would ideally be suitable for edge detection with a kernel size of (3, 3).

3.1.2.2 Gaussian blur or smoothening

Another method that reduces noise and details in an image is the Gaussian blur. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen. It uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. Values from this distribution are used to build a convolution matrix which is applied to the original image. Each pixel's new value is set to a weighted average of that pixel's neighborhood. The original pixel's value receives the heaviest weight (having the highest Gaussian value) and neighbouring pixels receive smaller weights as their distance to the original pixel increases. This results in a blur that preserves boundaries and edges better than other, more uniform blurring filters.

In our project, as shown in Figure 3.5., we tried to smoothen sample images (normal and covid) with a kernel size of (3, 3), (5, 5) or (7, 7).

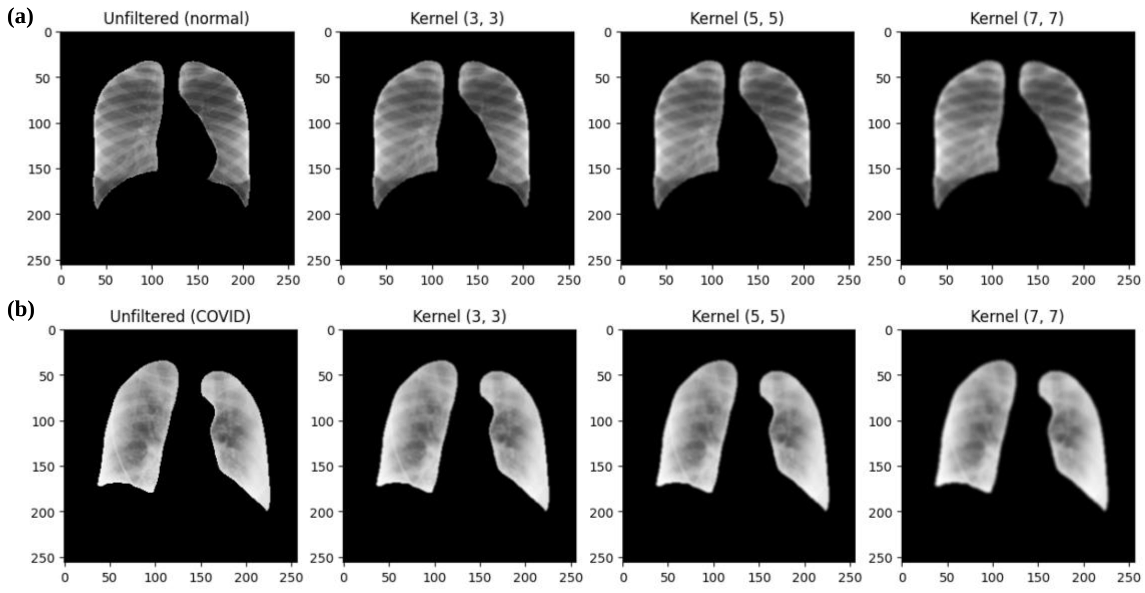


Figure 3.5. Gaussian blur of X-ray images (a) Representative lung area of a normal volunteer filtered with a kernel size (3, 3), (5, 5) or (7, 7) (b) Representative lung area of a COVID patient filtered with a kernel size (3, 3), (5, 5) or (7, 7). The scale signifies the width and height of the image in pixels.

As previously observed, kernels with a higher ksize significantly blurred the image. This reduced the noise but introduced shapes due to incomplete edge detection of the ribcage and other areas (Figure 3.6. (a, b)). These areas could be helpful in differentiating a normal and COVID lung. Therefore, we decided to use the kernel size (3, 3) with any filter that will be tested in our project. A Gaussian blur using a (3, 3) kernel followed by edge detection seemed best in reducing noise and detecting the haziness in the lung area of patients with COVID as structures (Figure 3.6. (b)). This could be a potential filtering technique that could be used to pre-process images for machine learning.

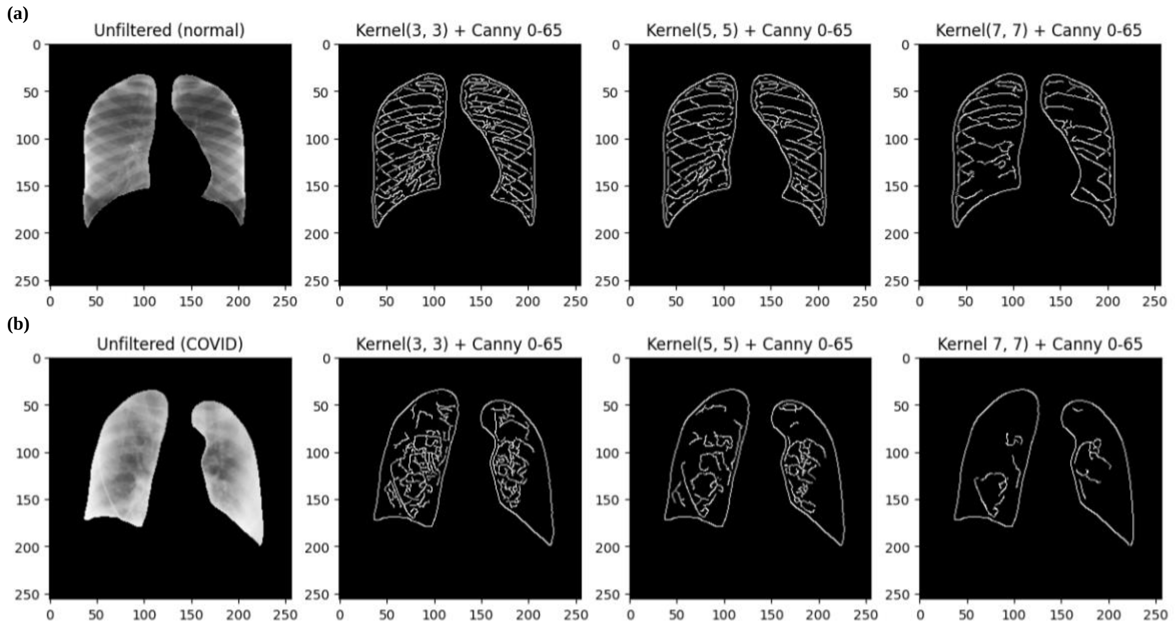

(a)
Unfiltered (normal)
Kernel(3, 3) + Canny 0-65
Kernel(5, 5) + Canny 0-65
Kernel(7, 7) + Canny 0-65
(b)
Unfiltered (COVID)
Kernel(3, 3) + Canny 0-65
Kernel(5, 5) + Canny 0-65
Kernel 7, 7) + Canny 0-65
0
50
100
150
200
250

Figure 3.6. Gaussian blur and Canny edge detection (a) Gaussian filtered X-ray image of a healthy individual followed by a Canny edge detection (threshold - min: 0, max: 65) (b) Gaussian filtered COVID X-ray image followed by a Canny edge detection (threshold - min: 0, max: 65)

3.1.2.3 Canny edge detection

The Canny edge detection algorithm was developed by John F. Canny in 1986. It is a multi-step process consisting of:

- Applying Gaussian smoothing (or other filters in our case) to the image to help reduce noise
- Computing the image gradients using the Sobel kernel
- Applying non-maxima suppression to keep only the local maxima of gradient magnitude pixels that are pointing in the direction of the gradient

- Defining and applying the minimum and maximum thresholds for Hysteresis thresholding

In our project, we detected the edges after each filtering step described in the pre-processing to visualize the impact of each filter respectively. We used a minimum threshold value of 0 and a maximum threshold between 50-100 initially. A maximum threshold of 75 and above did not completely detect the ribcage in normal (Figure 3.7. (a)). Additionally, in the case of COVID lungs, the haziness was not completely captured (Figure 3.7. (b)). Therefore, we decided to focus on edge detection with the maximum threshold range of 65 (see figure 3.6. (b)) for pre-processing.

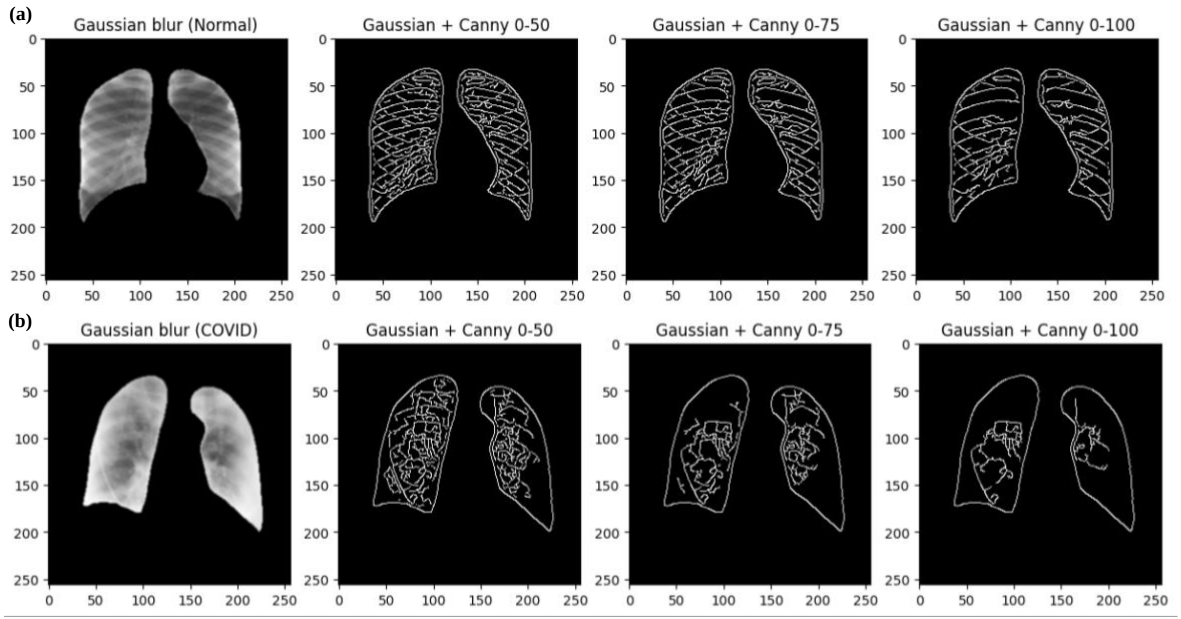


Figure 3.7. The Canny edge detection after a Gaussian blur. Representative lung area of (a) a normal volunteer (b) a COVID patient after a Gaussian blur (kernel size (3, 3)) and Canny threshold of (min: 0, max: 50, 75 or 100). The scale signifies the width and height of the image in pixels.

3.1.2.4 Erosion

This technique processes images based on the shapes in the image. It computes a local minimum over the area of a given kernel. As the kernel is scanned over the image, the minimal pixel value overlapped by it is computed and the image pixel under the anchor point is replaced with that minimal value. Visually, erosion erodes away the boundaries of the foreground object. Figure 3.8. shows a representative X-ray image of a normal individual with/without erosion and Canny edge detection. When we eroded this image with a (3, 3) kernel followed by Canny edge detection (min = 0, max = 65), we identified that the ribcage was thinner than that observed in other filters. However, like all other filtering techniques, erosion reduced noise and detected the crucial structures of the lung.

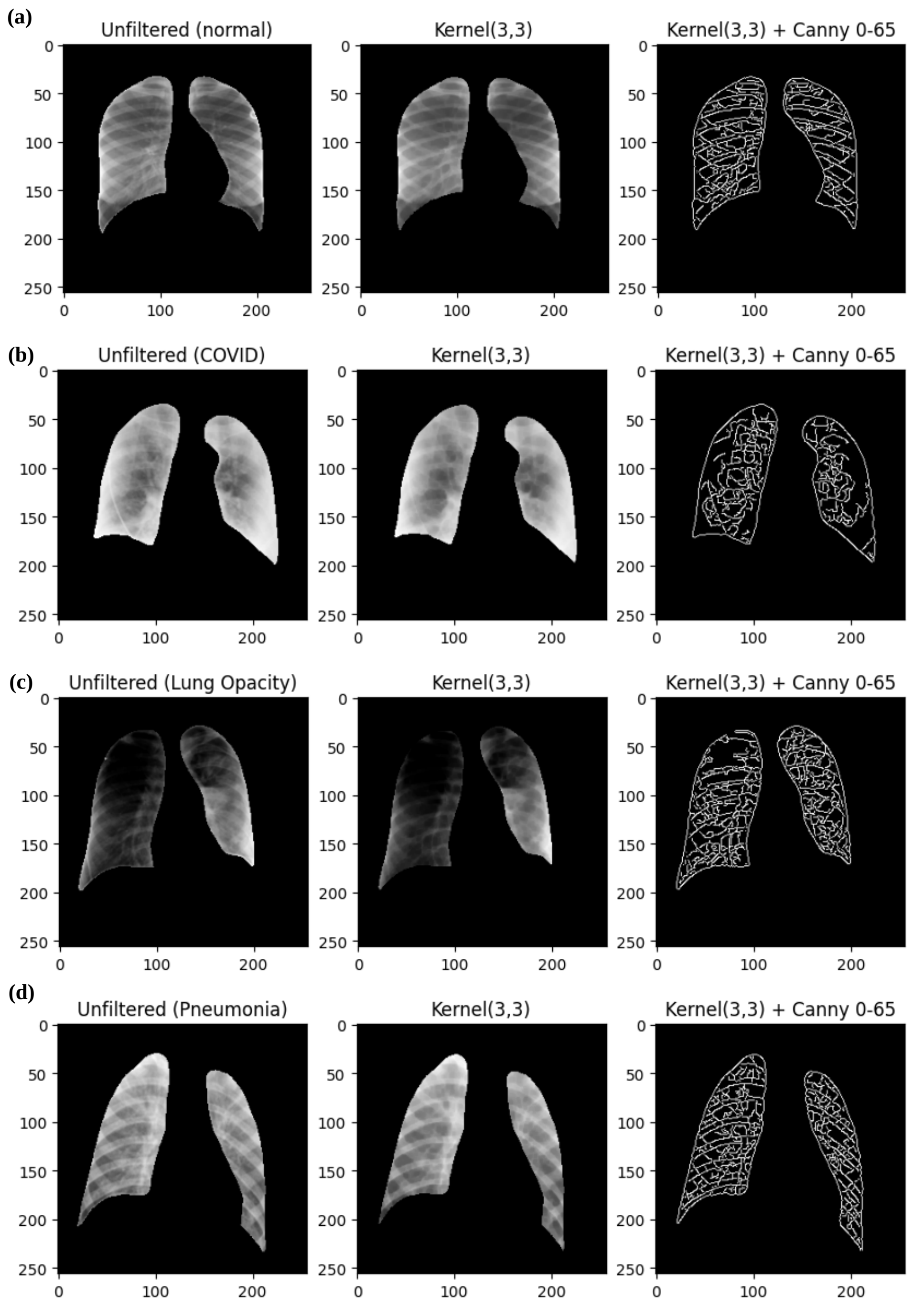


Figure 3.8. Erosion and Canny edge detection. (a) Representative raw or filtered lung area of normal volunteer with/without Canny edge detection (b) Representative COVID filtered lung area with or without erosion and Canny edge detection (c) Representative raw or filtered lung area of a person with an opaque lung and with/without Canny edge detection (d) Representative raw or filtered lung area of a person with viral pneumonia with/without Canny edge detection. The scale signifies the width and height of the image in pixels. The min and max threshold values for Canny edge detection was 0 and 65.

3.1.2.5 Laplacian filter

The Laplacian filter detects sudden intensity transitions in the image and highlights the edges. It convolves an image with a mask [0,1,0; 1,− 4,1; 0,1,0] and acts as a zero crossing detector that determines the edge pixels. We applied the Laplacian filter using a ddepth of 64F. As shown in Figure 3.9., this filter did not detect the edges well and we almost did not observe much in the lung area especially in case of COVID and hence, we did not explore or use this filter for further analysis.

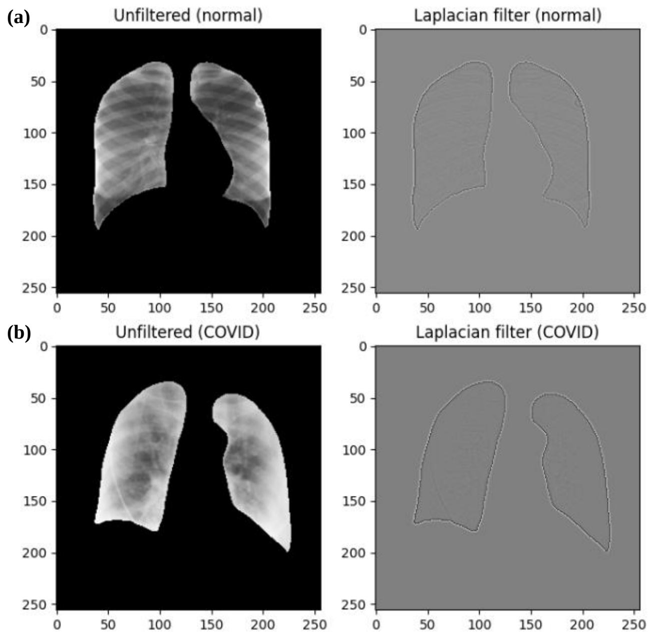


Figure 3.9. Laplacian filter. **(a)** Representative X-ray image of a normal individual with or without the Laplacian filter and **(b)** Representative X-ray image of a COVID patient with or without the Laplacian filter. The scale shows the size of the image (256*256) in pixels.

After extensively studying all filters, we decided to consider only one filter and the raw lung image for further modelling and analysis. The median blur, Gaussian blur and erosion provided desirable results with a (3, 3) kernel. However, we chose the Gaussian filter as it is widely used for X-ray pre-processing [1]. We would use filtered and raw lung images for basic classification models to see how feature extraction impacts the accuracy of the model. Only raw images would be used to train deep learning models as feature extraction is already a part of complex models such as CNN and Lenet.

3.2 Classical Modelling

Even though we work with images, with the aim to go into deep learning, running classical models is important for the sake of learning, and showing why deep learning techniques, such as CNN, are more suitable.

The classical models done on the whole images are Linear Regression, Support Vector Machine, Random Forest, KNeighbors Classifier (KNN), Decision Tree, Bagging, AdaBoost, XGBoost, and Voting. The accuracies and total training times for each model are shown in Figure 3.10. and Figure 3.11., respectively.

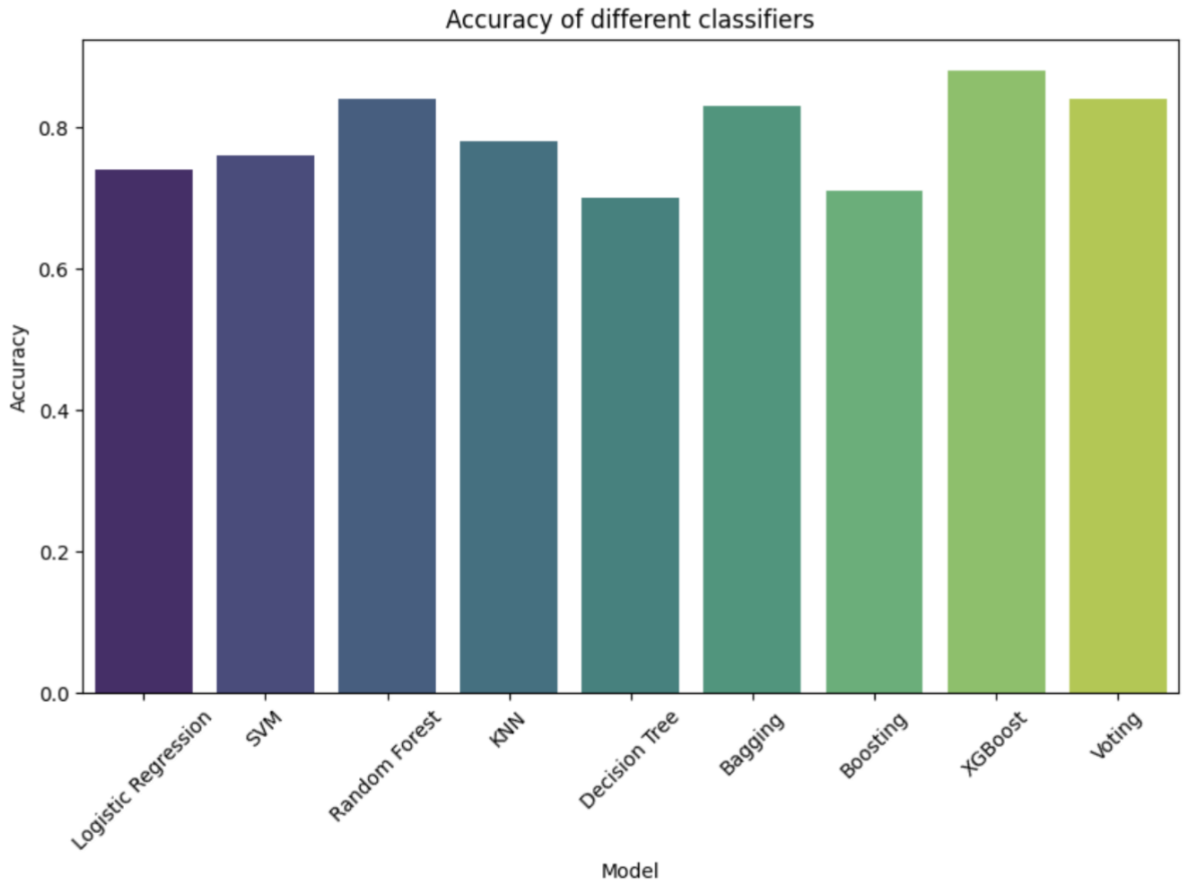


Figure 3.10. Accuracies of different Classification models trained on the whole X-rays

Before training KNN model, the number of neighbors was optimized, and it was obtained that the biggest accuracy is when the number of neighbors is 7. Voting Classifier was run with the following estimators: Random Forest Classifier, Bagging Classifier and XGB Classifier, since these three gave the best accuracies. Despite high accuracy, total training time for Voting Classifier was the highest. It is important to note that training time depends on the computer performances and will be different depending on the computer. However, all the classification models were run on the same computer, and comparing training times of each model makes sense at this point. This may be especially important when making decision with which model to continue to work.

We have run Stacking Classification Model as well. However, since the training time took more than 2 days, the training was stopped and the results were not obtained.

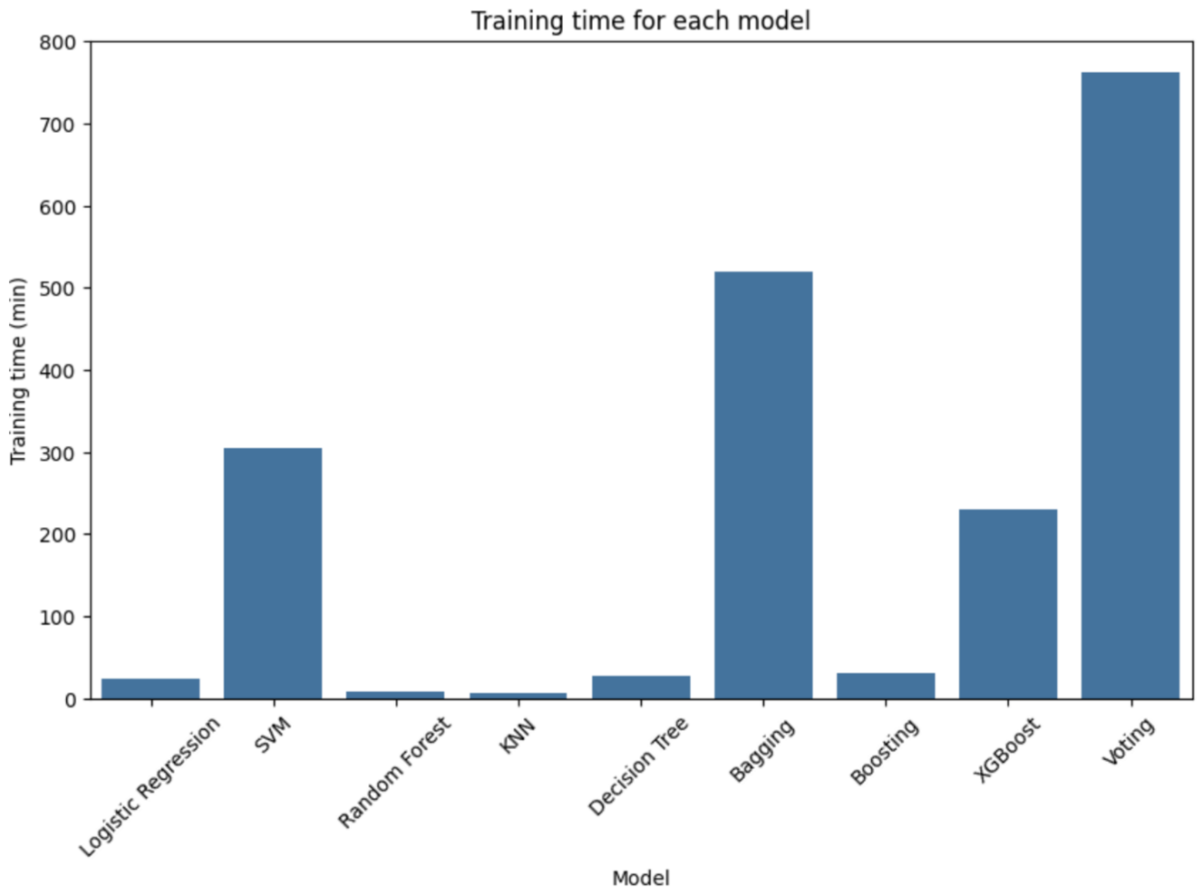


Figure 3.11. Total training time for each model trained on the whole X-rays

The best training time had Logistic regression, Random Forest, KNN, Decision Tree, and AdaBoost. Since Random Forest had one of the best accuracies and optimal training time, we will consider it for future trainings. Also, since Bagging and XGBoost have high accuracies and acceptable training time, we will consider these two models as well. Parameters used for these three models are give in Table 3.1

Table 3.1. Parameters for Random Forest, Bagging and XGBoost

Model	Parameter	Value
Random Forest	n_estimators	100
	random_state	123
Bagging	estimator	RandomForestClassifier()
	n_estimators	100
	random_state	123
XGBoost	objective	'multi:softmax'
	num_class	4
	scale_pos_weight	class_weights

We would like to emphasize that for each model, except for KNN and XGBoost, classical weighting was included and models were balanced. On the other side, KNN and XGBoost do not support class weights. KNN is a model with distance-based voting, and classifies a sample based on the majority vote of its k-nearest neighbors determined by distance metrics. Since this approach relies purely on proximity, it does not have mechanism to incorporate class weight. Additionally, introducing class weights into the distance calculations or the voting process would complicate the algorithm significantly. XGBoost focuses on sample weights through the boosting process. Gradient Boosting Mechanism optimizes the model by adding trees to minimize a loss function based on gradients, and it does not support class weights.
Furthermore, Bagging does not support class weights either, since involves generating multiple subsets of the training data by random sampling with replacement (bootstrap sampling). However, we have handled the class imbalance in Bagging by choosing RandomForestClassifier as an estimator.

In KNN and XGBoost, sample imbalance can be handled by oversampling or undersampling. When performing the RandomUnderSampler function from the "imblearn" library on the training set, the training set was reduced from 16,932 to only 4,208 rows. The resulting accuracy scores dropped significantly which is why we didn't continue to use random undersampling.

Table 3.2. Accuracy scores with undersampled training data

Classifier	Accuracy Score
RandomForest	0.79

AdaBoost	0.52
LogisticRegression	0.72
KNeighbors	0.71
SVC	0.78

We also attempted to use random oversampling. Unfortunately, when applying the RandomOverSampling function to the training set, the kernel crashed after several hours and due to time issues we didn't follow this strategy of balancing the training data any further.

Along with accuracies and training times, we have calculated Precision, Recall, and F1-Score for each category and each model for statistically weighted dataset of whole X-rays. We have choose to show only metrics for Random Forest, Bagging and Boosting on Figure 3.12., Figure 3.13., and Figure 3.14. For all the other models, information could be found in the notebooks.

Precision scores in classification models are a metric used to evaluate the performance of a model. Precision is calculated as the ratio of true positive predictions to the total number of predicted positive instances. High precision indicates that the positive predictions of the model are reliable and that the model is good in avoiding false positives.

Recall, or Sensitivity, or True Positive Rate, measure's the model's ability to correctly identify all positive instances, true positives and false negatives. Out of all the actual positive instances, how many did the model correctly identify, is the question Recall answers. High recall indicated that the model is good at capturing all positive instances, minimizing false negatives.

F1-Score combines precision and recall into a single metric that balances between them. High F1-Score indicates both high precision and high recall, reflecting a model that can reliably predict positive instances while also capturing a high proportion of them.

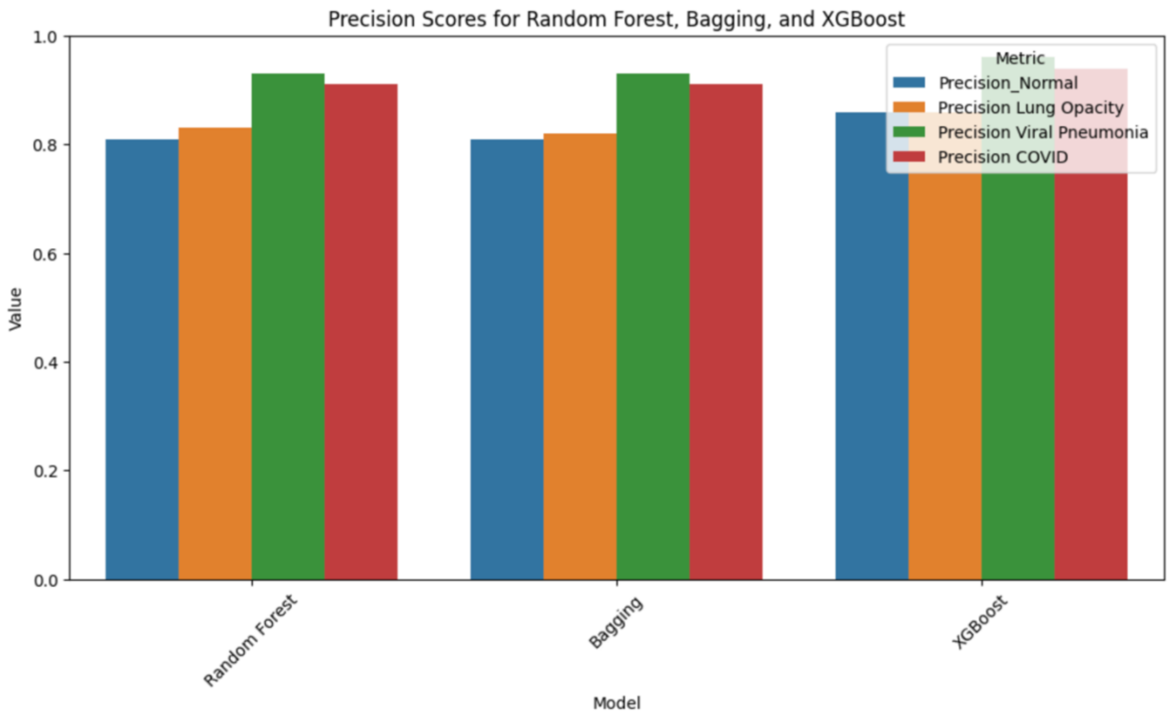


Figure 3.12. Precision Scores for each category in Random Forest, Bagging and XGBoost

By looking at the Figure 3.12., we can conclude that all the models have more or less similar precision scores. Precision Scores with 0.8 or higher, indicate good model predictability.

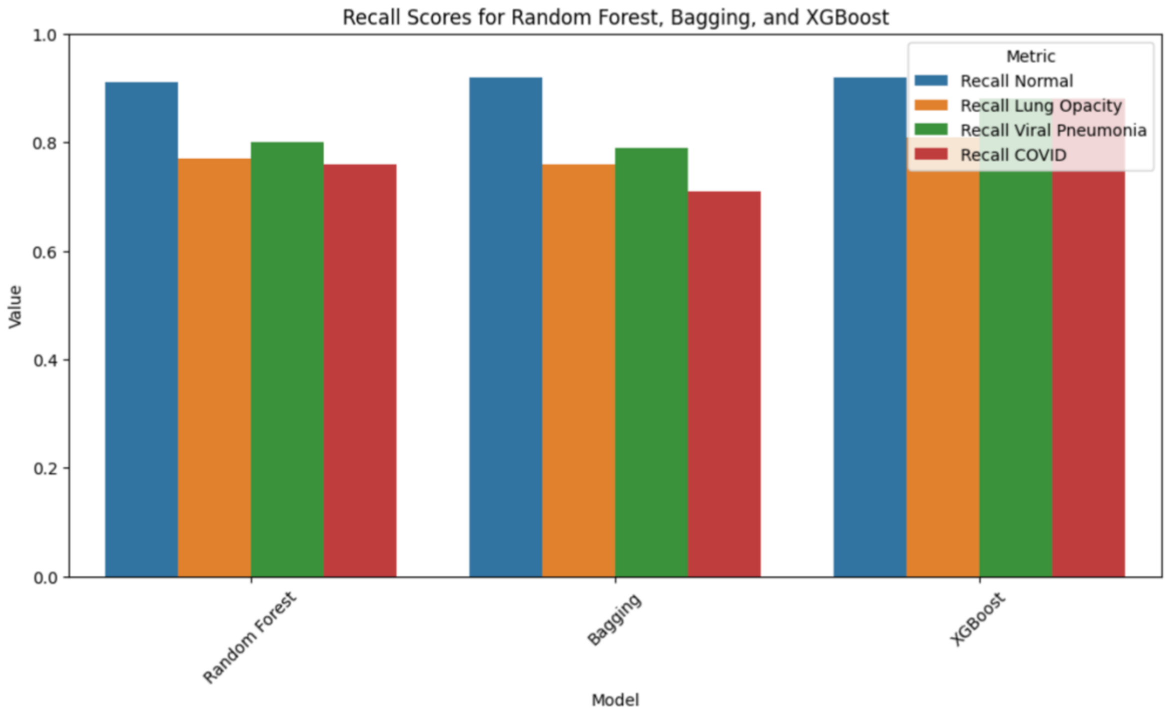


Figure 3.13. Recall Scores for each category in Random Forest, Bagging and XGBoost

It can be observed from Figure 3.13., that the highest Recall has the category **Normal**, which at the same time has the highest number of images.

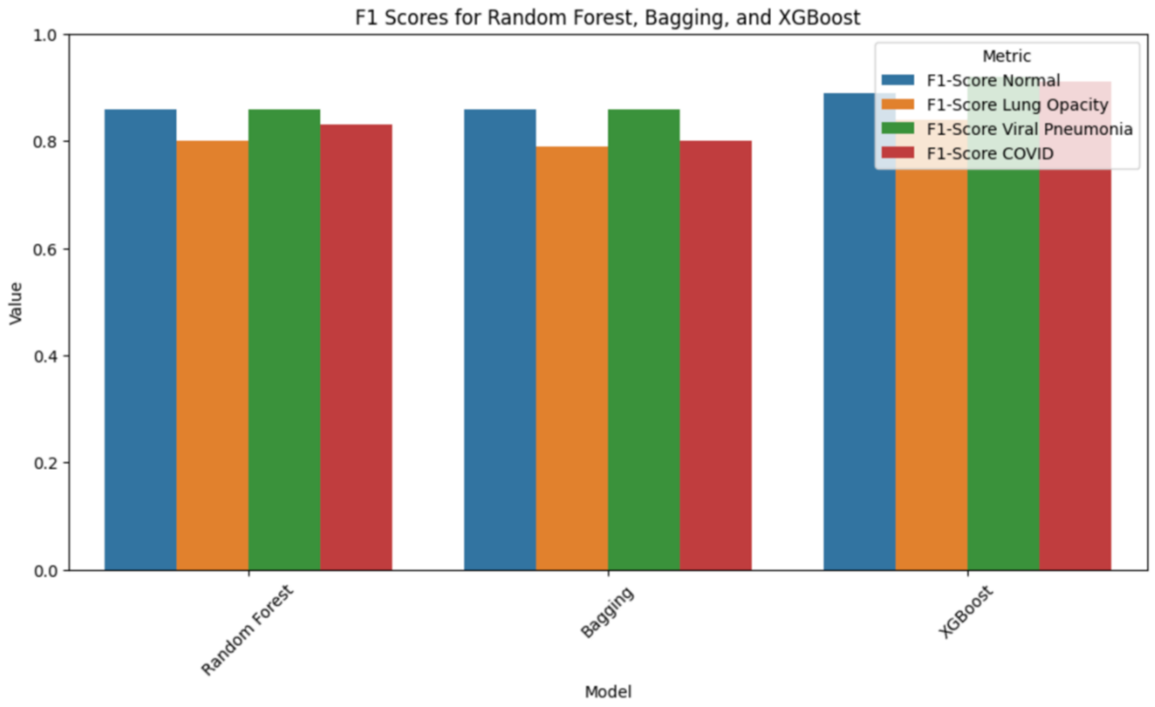
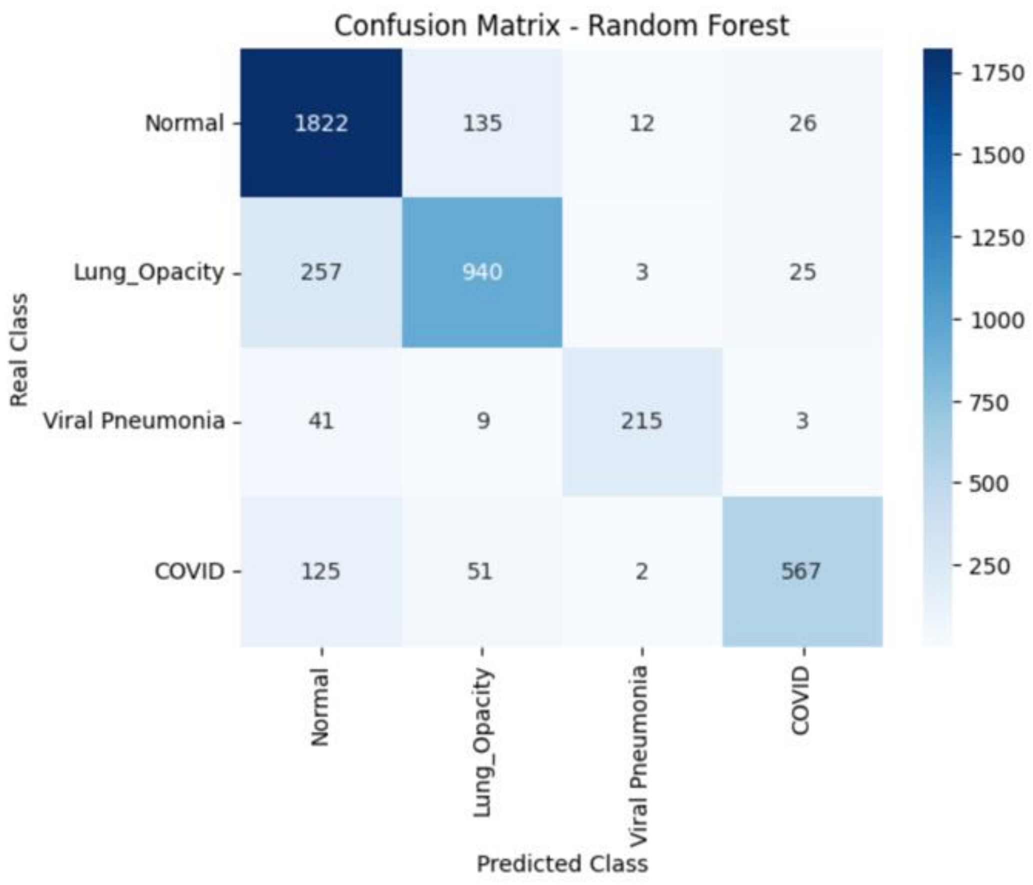


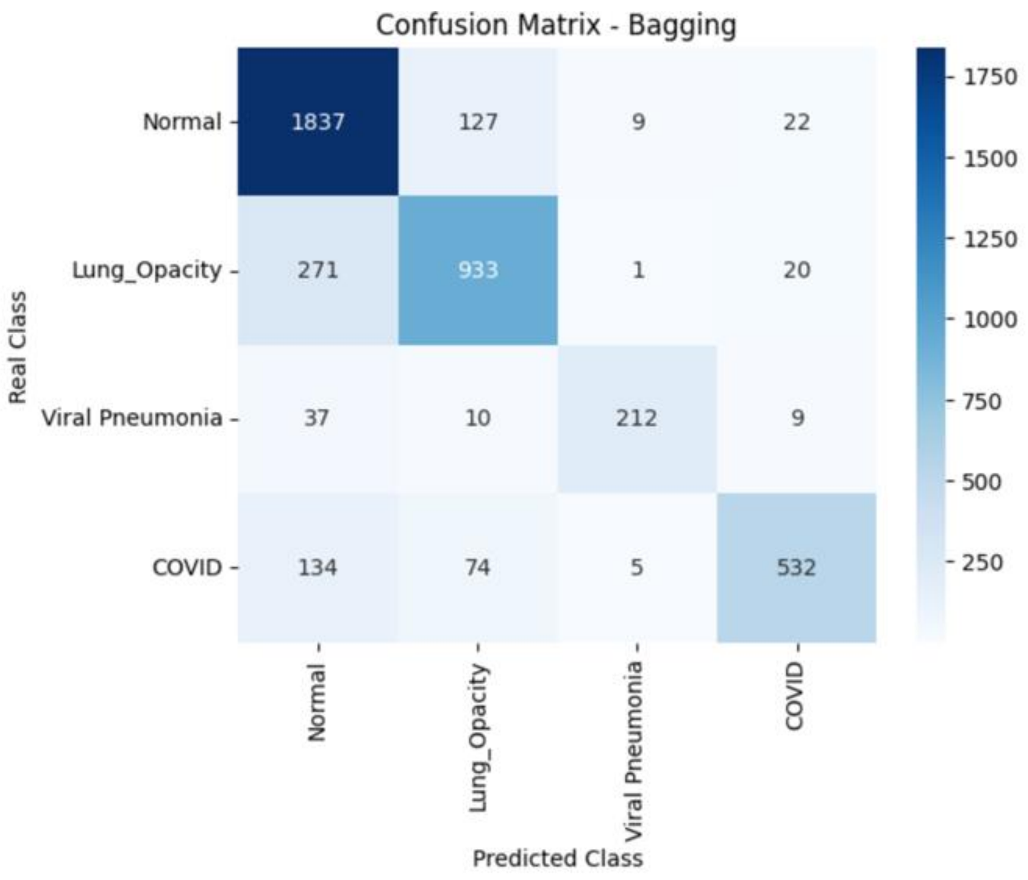
Figure 3.14. F1 scores for each category in Random Forest, Bagging and XGBoost

From Figure 3.14., we can observe that some of F1-Scores are higher than 0.8, suggesting that the model is accurately predicting both positive and negative instances for the class. It implies that the model is not only able to correctly identify most of the true positives but also has a low rate of false positives and false negatives. According to the F1-Scores, models that are performing the best are Random Forest, Bagging, XGBoost, and Voting (data could be found in notebooks), which is in correlation with previously showed accuracies.

A confusion matrix summarizes the performance of a classification model by presenting the counts of true positive, true negative, false positive, and false negative predictions made by the model. It provides a detailed breakdown of how well the model is performing in terms of classifying instances into different classes. Confusion matrix is used as a diagnostic tool, since it helps in understanding where the model is making errors, false positives and false negatives. It also provides a more detailed assessment beyond simple accuracy, especially useful in imbalanced datasets where classes are not equally represented. Furthermore, it helps in selecting an appropriate decision threshold based on the specific needs. A confusion matrix is a fundamental tool in evaluating the performance of a classification model, offering insights into its performance across different classes.
In Figure 3.15. are summarized all confusion matrices for the models performed.



Confusion Matrix - Random Forest
Real Class
Predicted Class
Normal
Lung_Opacity
Viral Pneumonia
COVID
1822
135
12
26
257
940
3
25
41
9
215
3
125
51
2
567
1750
1500
1250
1000
750
500
250



Confusion Matrix - Bagging
Real Class
Predicted Class
Normal
Lung_Opacity
Viral Pneumonia
COVID
1837
127
9
22
271
933
1
20
37
10
212
9
134
74
5
532
1750
1500
1250
1000
750
500
250

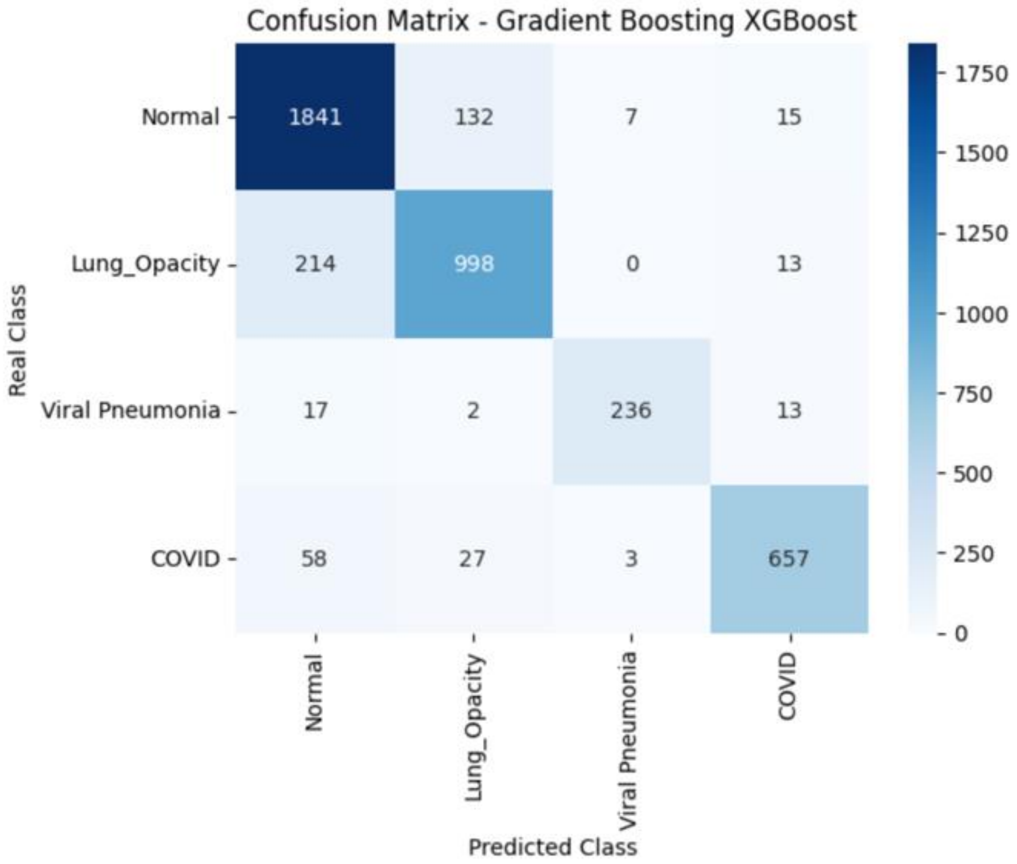


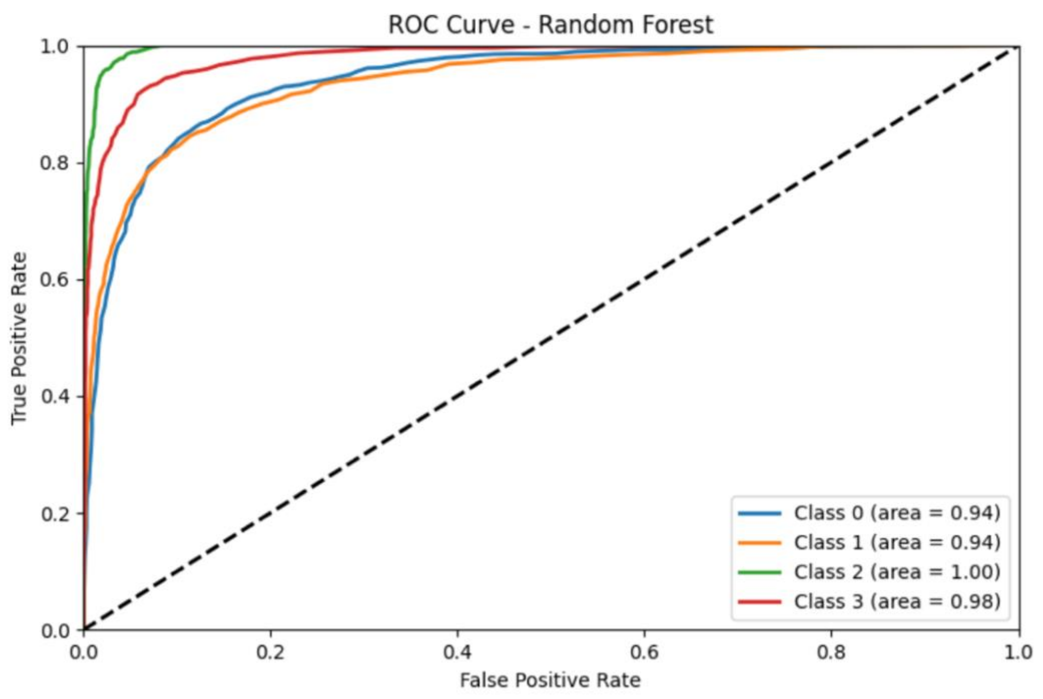
Figure 3.15. Confusion Matrices for different Classification models: Random Forest, Bagging and XGBoost

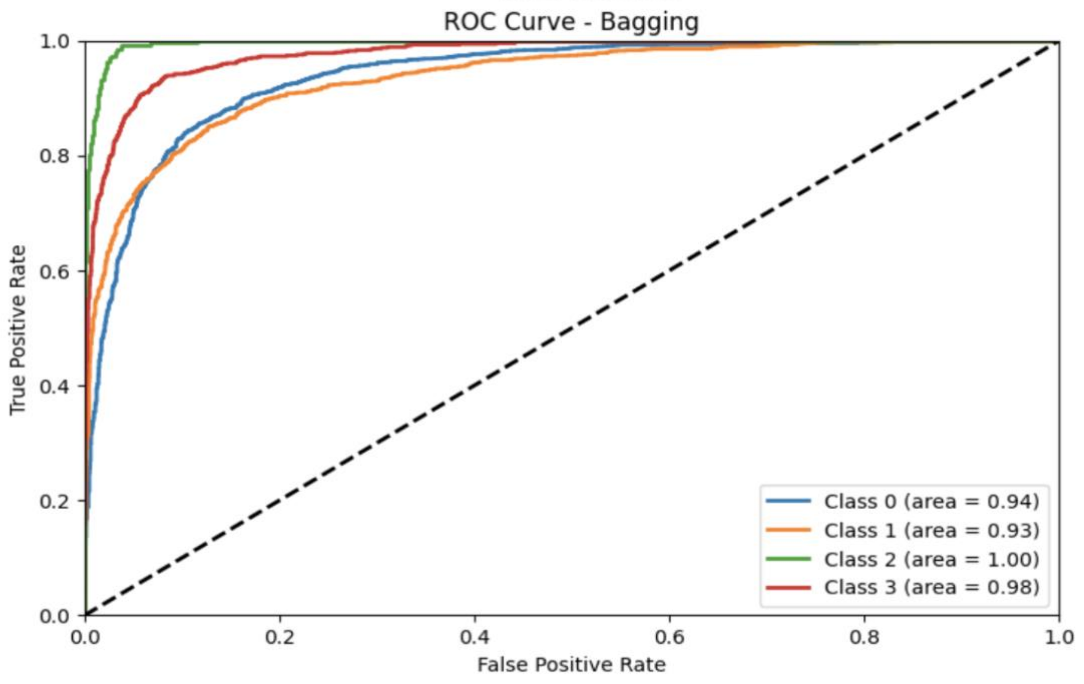
From the presented confusion matrices, we can conclude that in the case of all presented models, the biggest number of true positives always have category ***Normal***, while the lowest number of true positives always have category ***Viral Pneumonia***.

The Receiver Operating Characteristic (ROC) Curve is a graphical representation of a classifier's performance across different threshold settings, while it plots the True Positive Rate (TPR) against the False Positive Rate (FPR). The TPR (Sensitivity or Recall) is the proportion of actual positives correctly identified by the model, and FPR is the proportion of actual negatives incorrectly identified as positives. The area under the ROC Curve (AUC-ROC) is a value that summarizes the performance of the classifier. An AUC of 1 indicates a perfect classifier, while an AUC of 0.5 indicates a model no better than random guessing.

Precision-Recall (PR) Curve is a graphical representation that focuses on the performance of a classifier for the positive class, and it plots precision against recall for different threshold values.

The ROC and PR curves for Random Forest, Bagging and XGBoost are presented in figures below.





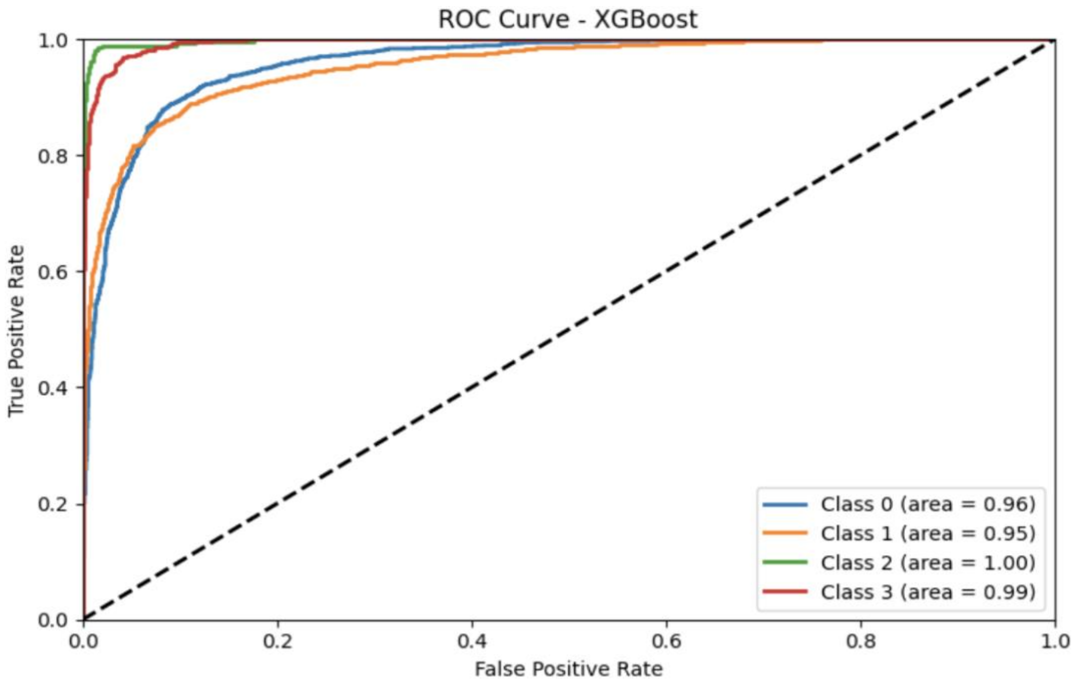
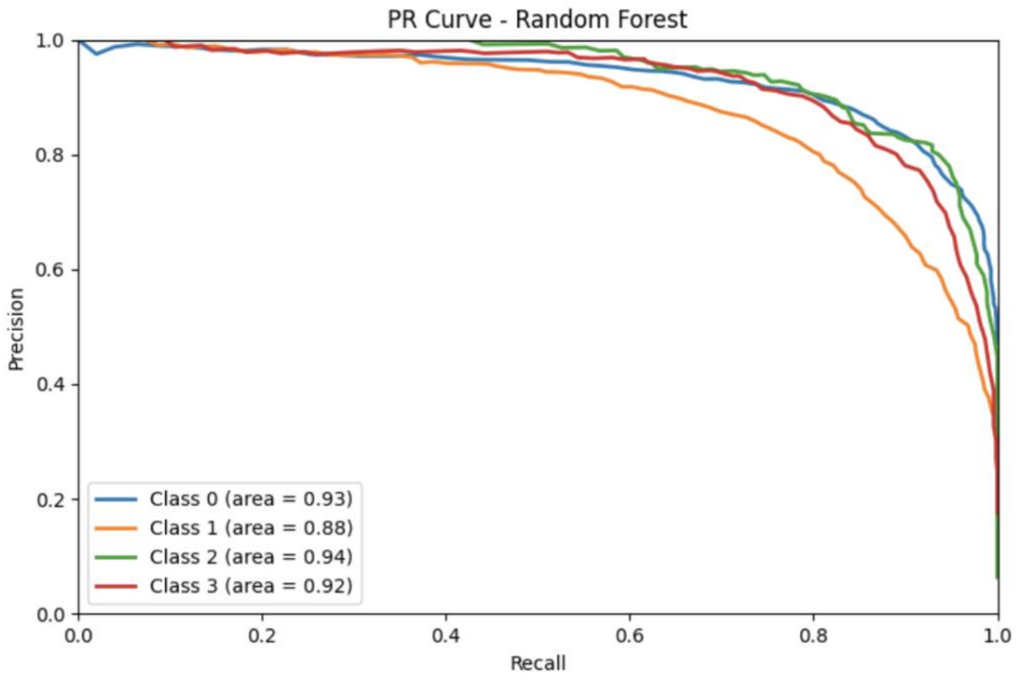
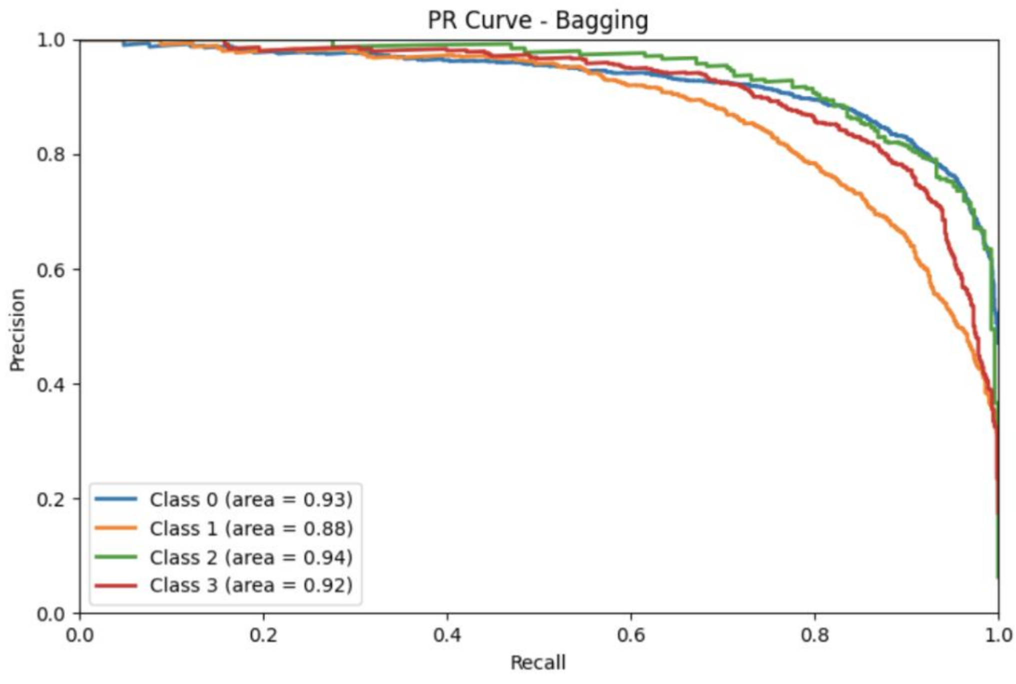


Figure 3.16. ROC Curves for all categories (Normal – 0, Lung Opacity – 1, Viral Pneumonia – 2, COVID – 3) obtained in Random Forest, Bagging and XGBoost

We can observe that for all categories in these three models, AUC-ROC are higher than 0.9 and close to 1, suggesting that all three classifiers are suitable.





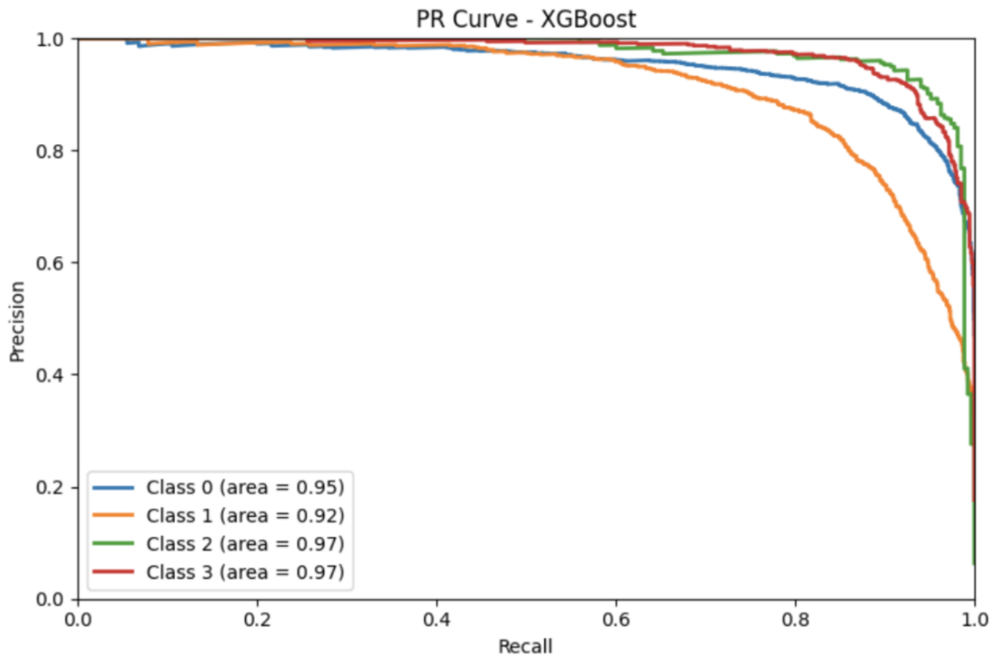


Figure 3.17. PR Curves for all categories (Normal – 0, Lung Opacity – 1, Viral Pneumonia – 2, COVID – 3) obtained in Random Forest, Bagging and XGBoost

As can be observed, the AUC-PR confirm previously obtained results, suggesting that Random Forest, Bagging and XGBoost have good predictability for the dataset of whole X-rays.

In Figure 3.18., we can see the common misclassified images for all classical models performed.

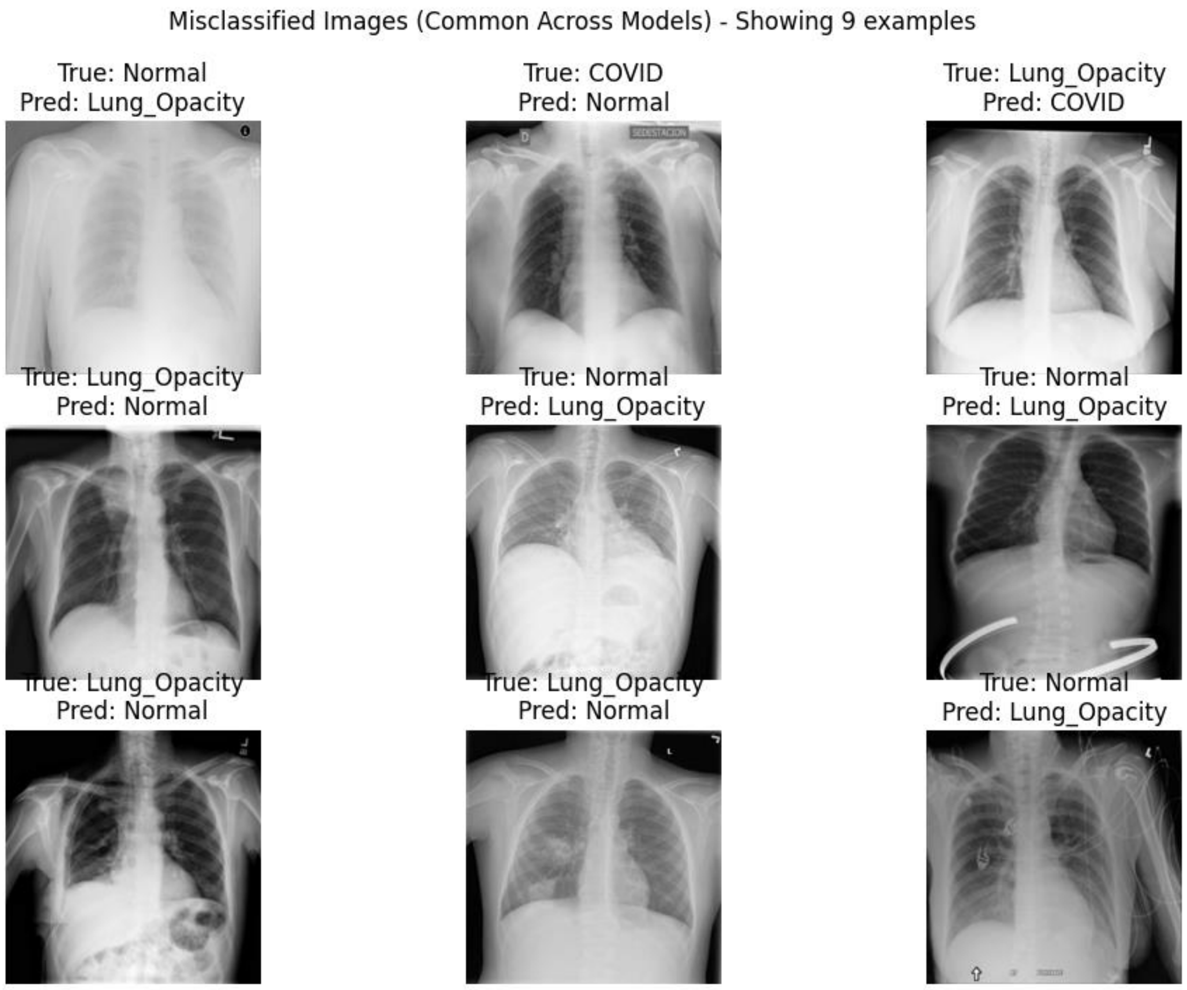


Figure 3.18. Common misclassified images across all performed models

As seen in the above presented, the models with highest accuracies, best performances, and acceptable training times are RandomForest, Bagging and XGBoost. Thus, we tried to optimize these three models before running them on the preprocessed data. However, the codes have never finished, since one optimization took more than two days. This is the reason why we decided to skip this step and just continue with running the three best models on two datasets: 1) dataset of images presenting Region of Interest (ROI), and 2) dataset of images presenting ROI but with filters. The datasets were statistically weighted to avoid disbalance and class weight was included in Random Forest and Bagging, while it is not possible for XGBoost.

Model Accuracies for raw Region of Interest (only lungs) are shown in Figure 3.19. As can be seen, the accuracies are slightly lower than it was the case with the whole X-rays. Figures 3.20. – 3.22. show Precision, Recall, and F1-Scores for each category. These values are lower than it was the case with whole X-rays, suggesting lower predictability of chosen models.

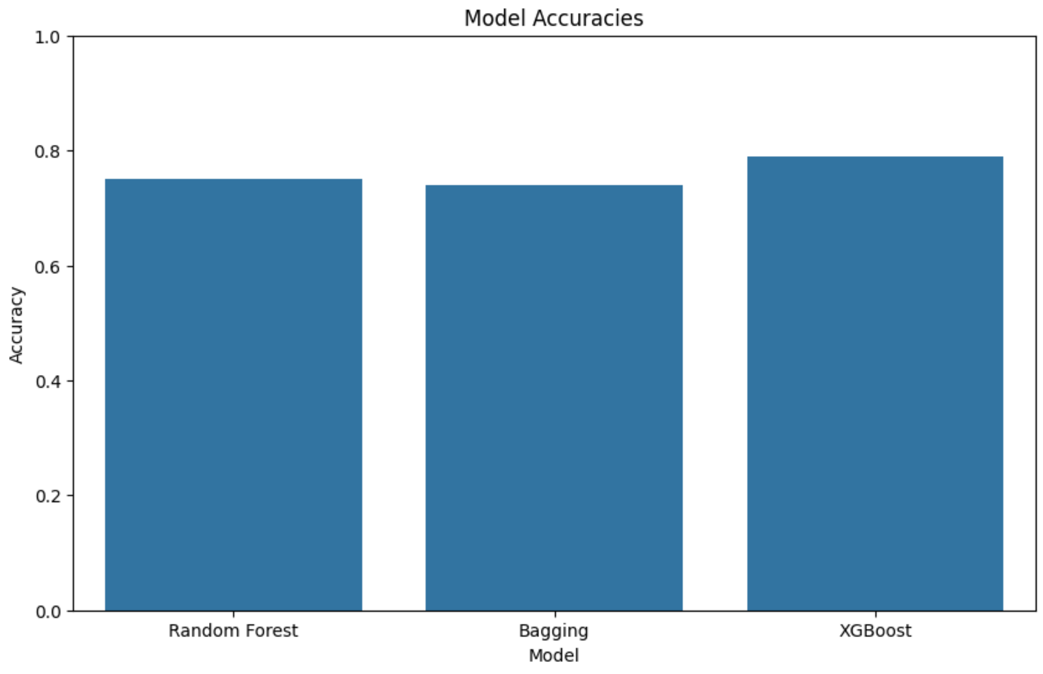


Figure 3.19. Accuracies of Random Forest, Bagging and XGBoost trained on ROI (lung area)

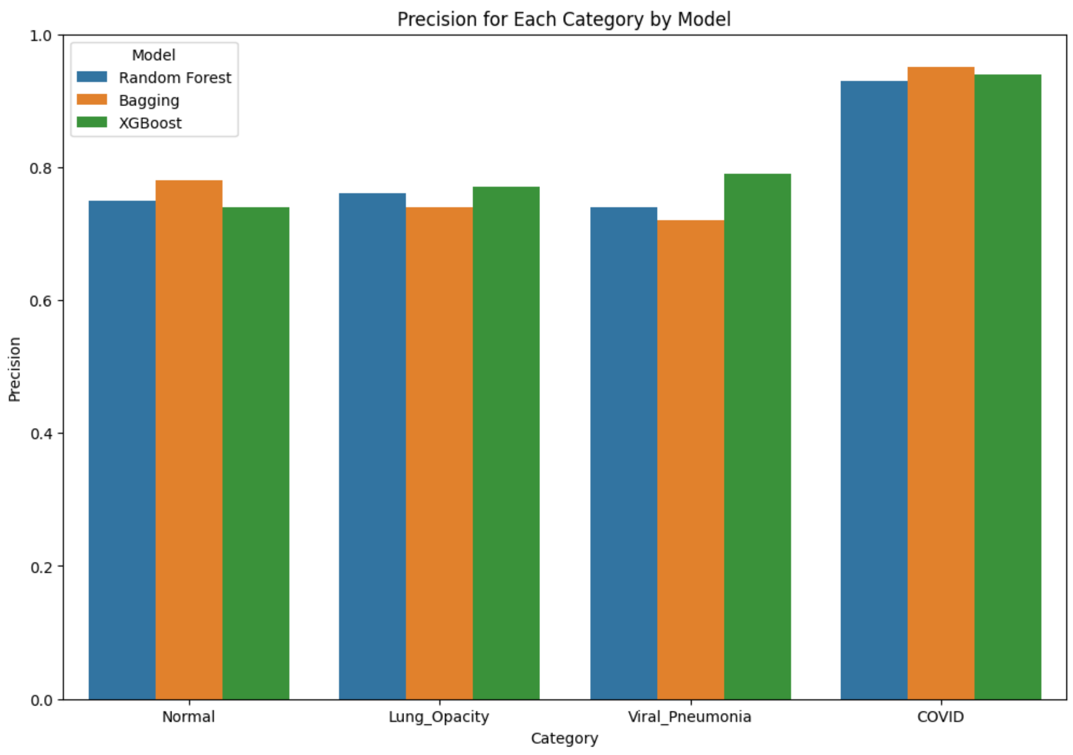


Figure 3.20. Precision Score for each category obtained in Random Forest, Bagging and XGBoost on ROI (lung area)

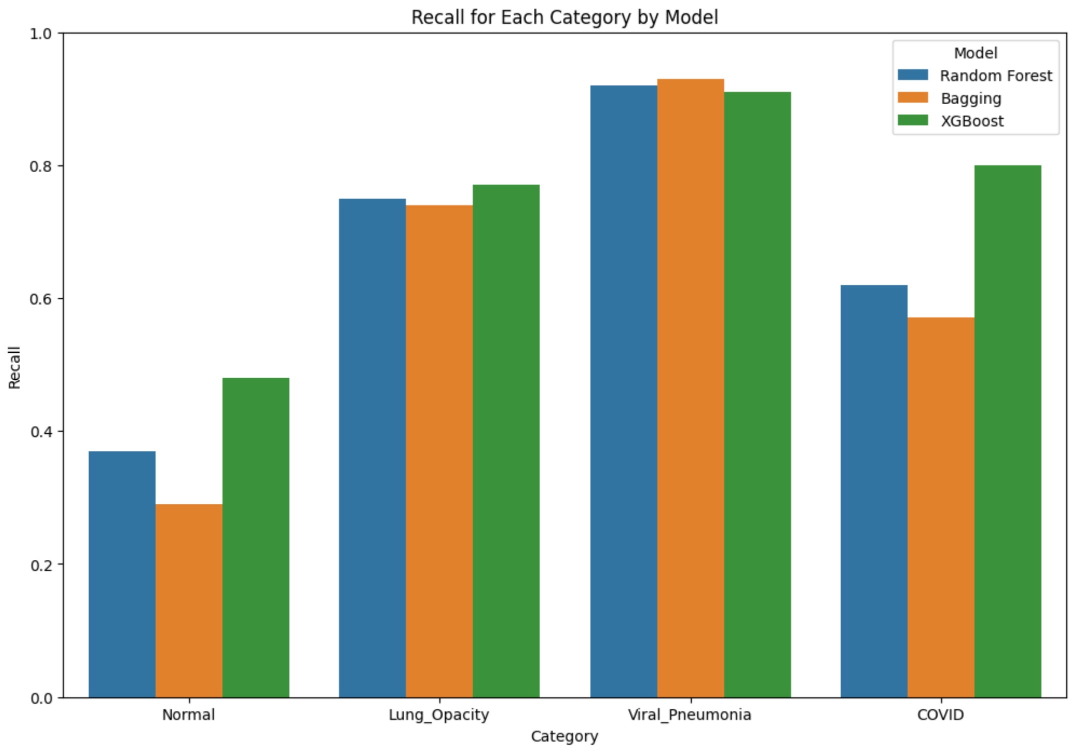


Figure 3.21. Recall for each category obtained in Random Forest, Bagging and XGBoost on ROI (lung area)

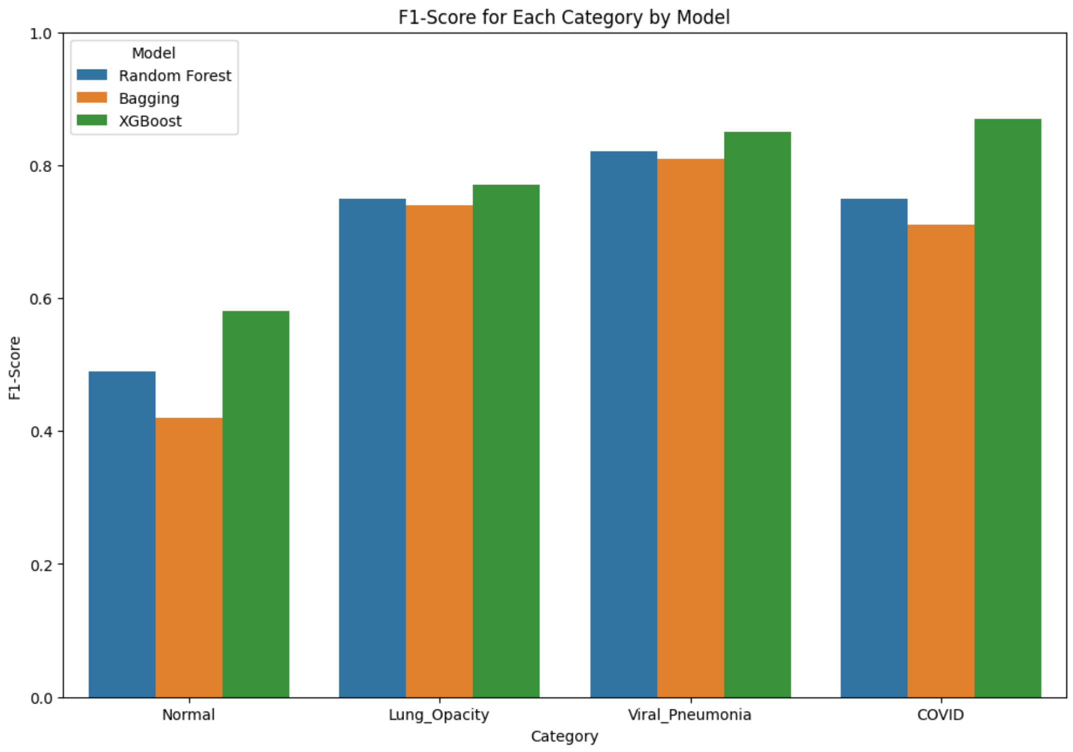
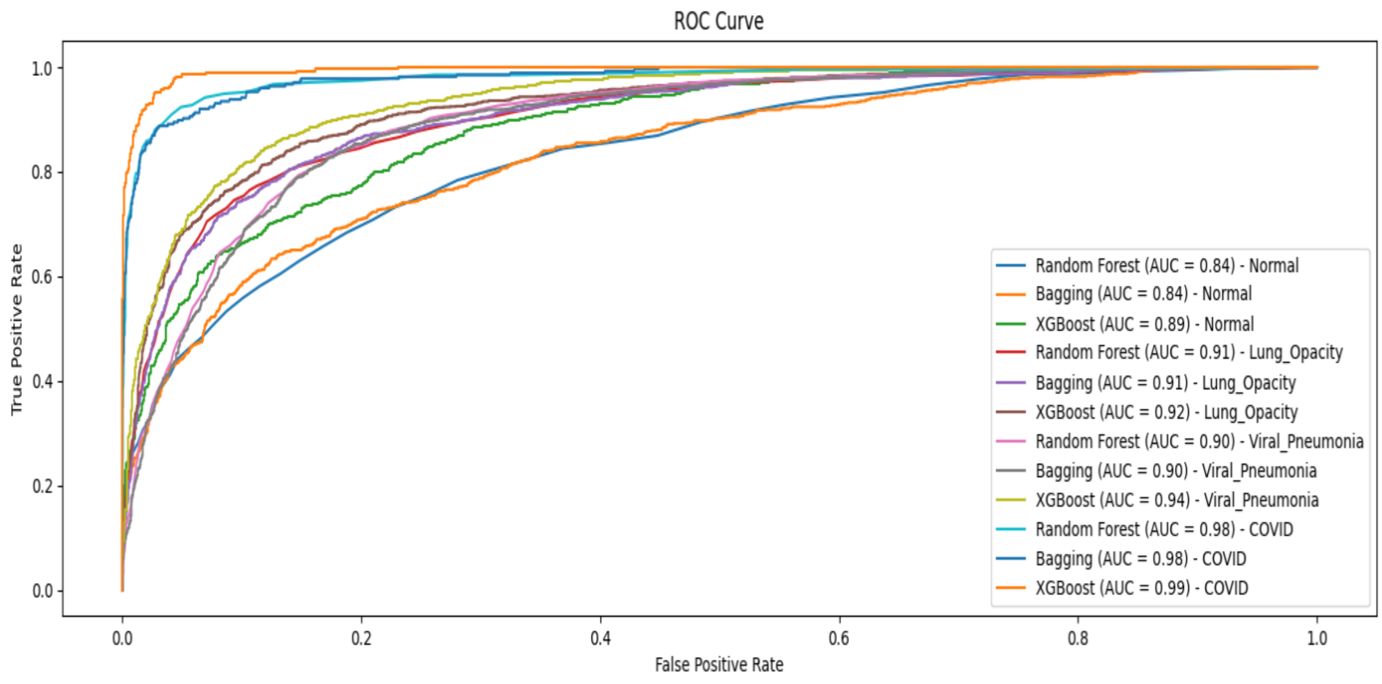


Figure 3.22. F1-Score for each category obtained in Random Forest, Bagging and XGBoost on ROI (lung area)



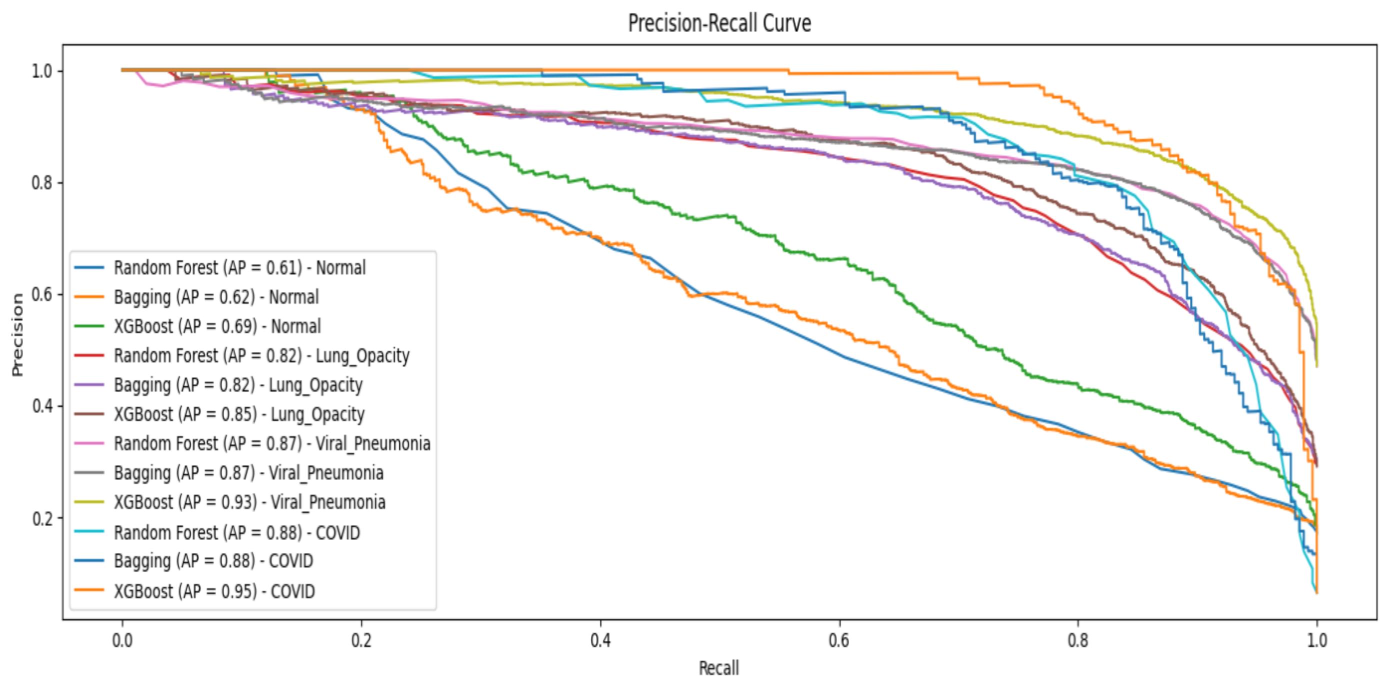


Figure 3.23. ROC Curve and PR Curve for each category obtained with Random Forest, Bagging and XGBoost on ROI (lung area)

AUC-ROC for Bagging and XGBoost are high and comparable to the AUCs obtained with the whole X-rays. However, the AUCs for Random Forest are slightly lower. This suggest that Bagging and Boosting may be suitable for our dataset. Areas under PR curves are lower as well, which is in accordance with data shown in Figures 3.19. – 3.22. Model Accuracies for filtered Region of Interest (only lungs) are shown in Figure 3.24. The obtained accuracies are lower than it was the case with the whole X-rays and in the case of raw ROI. Figures 3.25. – 3.27. show Precision, Recall, and F1-Scores for each category. These values are lower than it was the case with whole X-rays and raw ROI, suggesting lower predictability of chosen models.

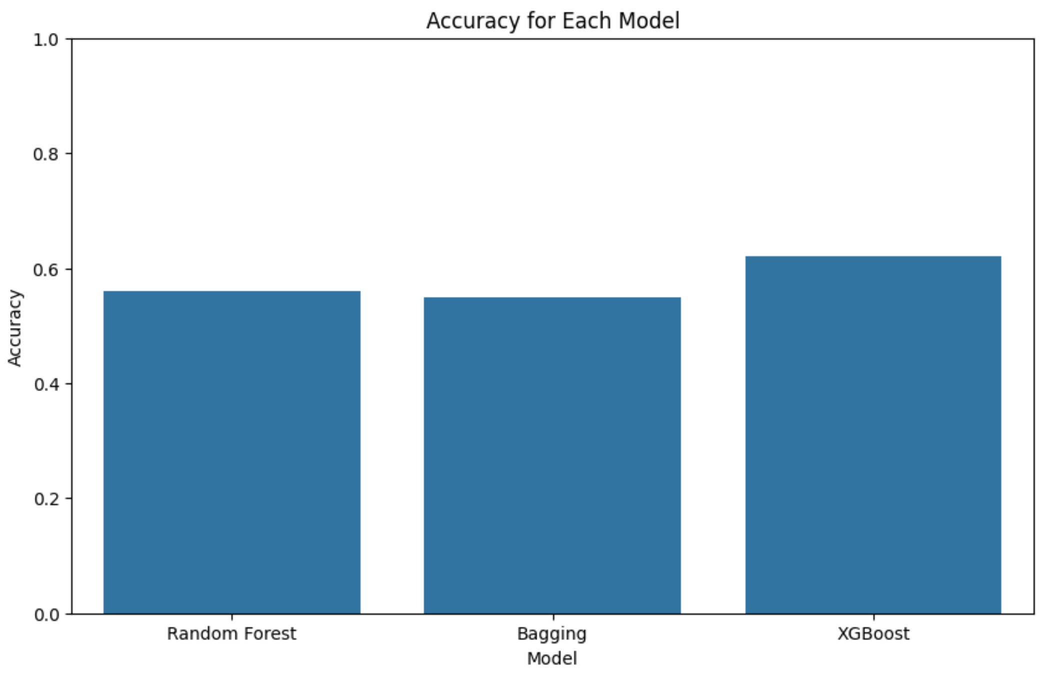


Figure 3.24. Accuracies of Random Forest, Bagging and XGBoost trained on filtered ROI (lung area)

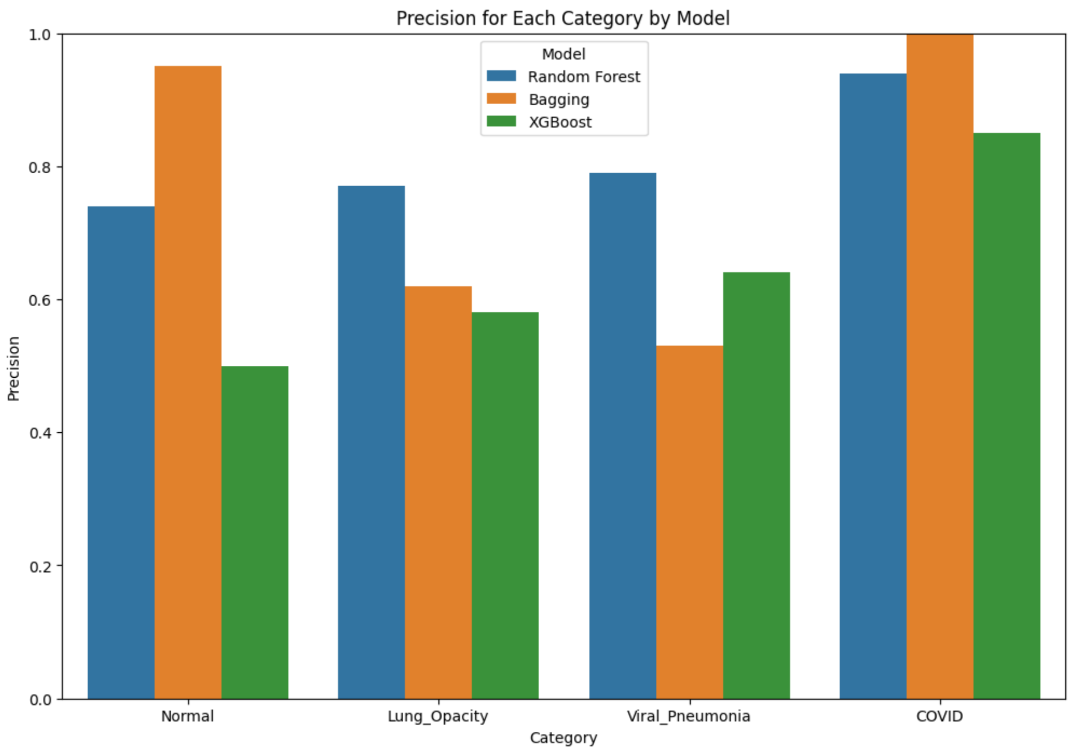


Figure 3.25. Accuracies of Random Forest, Bagging and XGBoost trained on ROI (lung area)

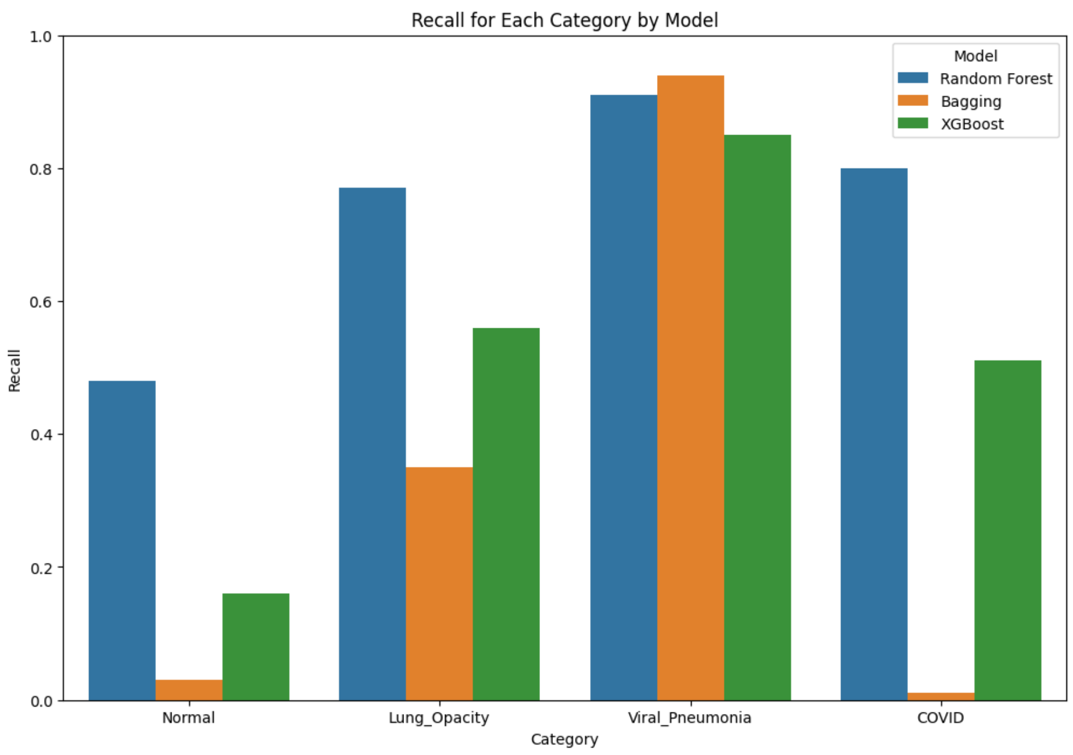


Figure 3.26. Recall for each category obtained in Random Forest, Bagging and XGBoost on filtered ROI (lung area)

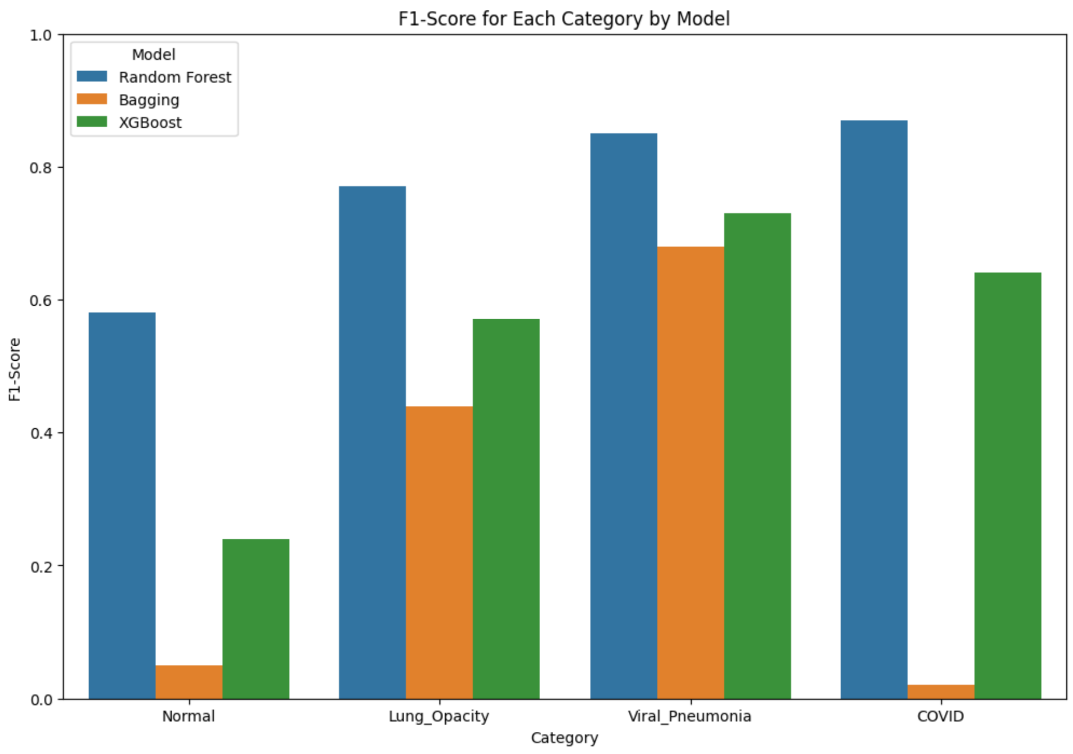


Figure 3.27. F1-Score for each category obtained in Random Forest, Bagging and XGBoost on filtered ROI (lung area)

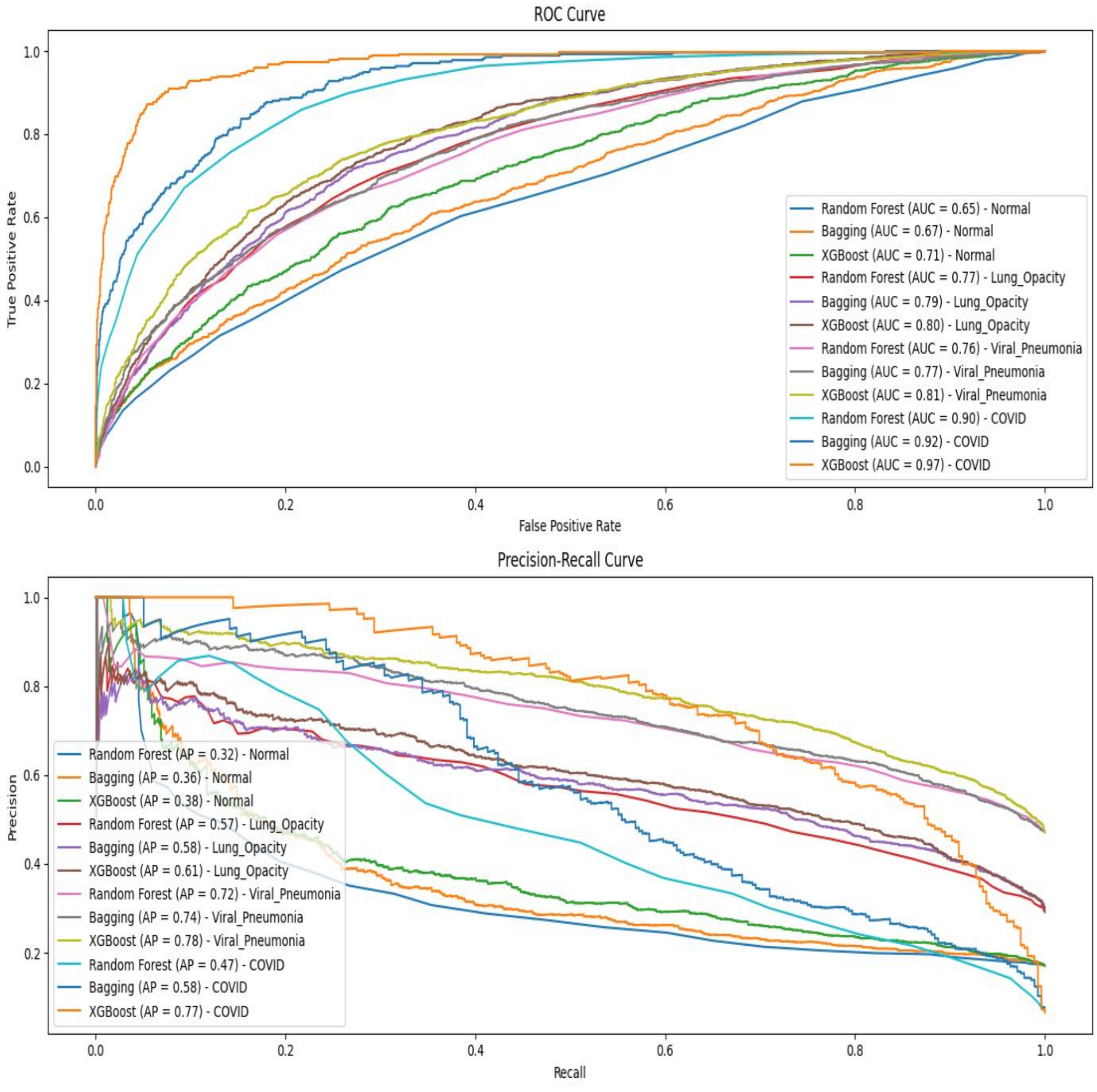


Figure 3.28. ROC Curve and PR Curve for each category obtained with Random Forest, Bagging and XGBoost on filtered ROI (lung area)

For all the metrics, fluctuations can be observed when models were trained on dataset of filtered ROI (lung area). In the case of ROC and PR curves, the AUCs and Aps are much lower than it was the case with raw ROI and whole X-rays. This suggest that these models may not be suitable for the dataset of filtered ROI.

In the next chapter we will present our Deep Learning models.

3.3 Deep Learning

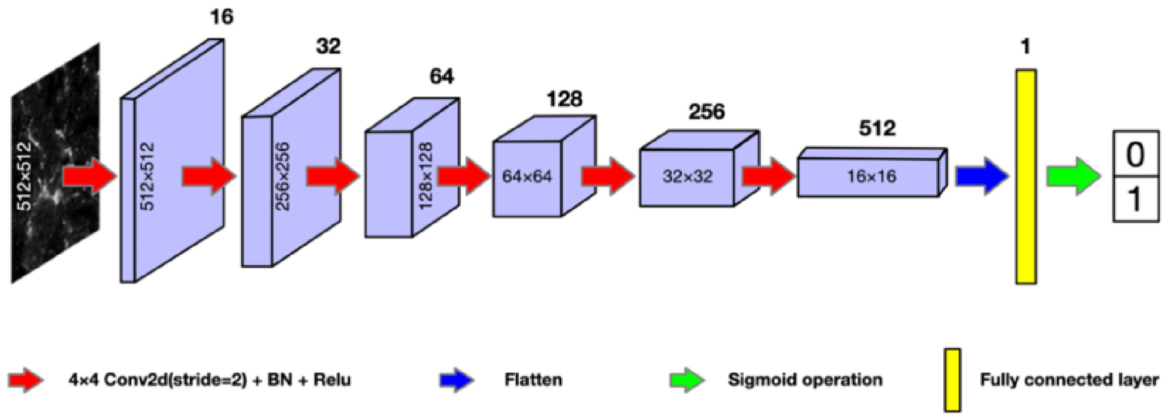
3.3.1. Introduction

Deep Learning (DL) is a subset of machine learning that involves neural networks with many layers (deep networks). It is characterized by its ability to automatically learn hierarchical representations of data. The evolution of DL has significantly advanced the field of image processing, enabling machines to perform tasks such as image classification, object detection, and image generation with high accuracy.

Neural Networks are one of many methods of DL and excel at handling topological data such as images. Neural networks are computational models inspired by the human brain, composed of layers of interconnected nodes (neurons). These networks transform input data through a series of weighted connections and activation functions to produce an output.

Convolutional Neural Networks (CNN) are specifically designed to handle grid-like data structures. They consist of multiple layers, primarily including convolutional layers, pooling layers, and fully connected layers. Lower layers capture basic features like edges and textures, while deeper layers capture more complex patterns and object parts.

By using small filters, CNNs exploit spatially local correlations in data, reducing the number of parameters compared to fully connected networks. This makes CNNs computationally efficient and suitable for large-scale image processing tasks. Scalability of CNNs are given by adapting the depth (number of layers) which allows the model to handle increasingly complex functions.



The core functional layers of a CNN encompass Convolutional, Pooling and Dense Layers:

- **Convolutional Layers**:
  - Apply convolution operations to the input image using filters (typically 3x3).
  - Filters slide over the image to produce feature maps that highlight specific patterns like edges, textures, and shapes.

- **Pooling Layers**:
  - Reduce the spatial dimensions of feature maps, preserving the most important information while reducing computational complexity.
  - Common pooling techniques include max pooling (taking the max of a striding window) and average (taking the mean of a striding window) pooling.

- **Fully Connected Layers (Dense Layers)**:
  - After several convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers.
  - These layers flatten the feature maps and connect every neuron to every neuron in the next layer, similar to traditional neural networks.

- **Activation Functions**:
  - Introduce non-linearity into the model, allowing it to learn complex patterns.
  - ReLU (Rectified Linear Unit) is the most commonly used activation function in CNNs. For the final Dense layer that predicts the classification a softmax or sigmoid function is applied. Since its upbringing the field of Convolutional Neural Networks has seen some developments in the usage of layers, types of layers and other techniques. Down below we introduce three commonly known milestone architectures which we will employ for our classification.

LeNet-5

Developed in 1998 LeNet-5 is one of the earliest CNN architectures. It was primarily designed for handwritten digit recognition and demonstrated the potential of Convolutional Neural

Networks (CNNs). LeNet-5 consists of seven layers: two convolutional layers, two pooling layers, and three dense layers.

AlexNet

Developed in 2012 AlexNet marked a significant breakthrough in the field of computer vision. AlexNet consists of five convolutional layers, followed by three fully connected layers. It employs ReLU activation functions, dropout for regularisation, and max-pooling layers. The use of ReLU activation functions addresses the vanishing gradient problem and the dropout layers prevent overfitting.

VGGNet

Developed in 2014 VGGNet emphasises simplicity and depth in network architecture design. VGGNet is known for its use of very small (3x3) convolution filters stacked in deep layers. The approach of using small convolution filters in deeper networks allowed for more complex features to be learned without a dramatic increase in the number of parameters.

LMA3P

The project groups approach of patching together a sensible selection of layers fit to the very challenge at hand.

3.3.2. Data preparation

In order to train our CNNs we prepare the image dataset in the following way

1. Resize images (224 x 224) while batching them into buckets of 32.
2. Split into 3 datasets: Training (80%), validation (10%) and test (10%).
3. Calculate the class weights on the training dataset for later user
4. Apply data augmentation techniques to the training dataset: Rotation (18°), Zoom(5%) and Horizontal Flip

3.3.3. Model Definition

The definition of the models can be seen further below at the end of this section. All models are being trained with the following parameters:

Optimizer: Adam
Loss function: Categorical Crossentropy
Metrics: Accuracy

Epochs: 25
Class Weights: Class weights as determined above

First we started out with LMAP3, for which we tried 3 different approaches and then took the best approach into the VGGNet and AlexNet:

1. LMAP3:
2. LMAP3:
3. LMAP3:
4. AlexNet: Full images with data augmentation
5. VGGNet:Fullimageswithdataaugmentation

Full images without data augmentation
Full images with data augmentation
Only lung section of images with data augmentation

Then we let the trained models predict the classes of our test dataset, which it had never seen before. The table below shows the overall accuracy as well as the average computation time per epoch

To see how the best model (B) and worst model (C) perform in more detail we take a look at the training progress as well as the respective confusion matrices.

Table 3.3. Accuracies and Time/epochs for different deep learning models

	A	**B**	**C**	**D**	**F**
Accuracy	0.90	0.91	0.83	0.85	0.75
Time / epoch	22.1 s	34.4 s	35.9 s	24.2 s	300 s

The confusion matrices shows the percentage of predictions per row, i.e. per real class. A value of 91.73% means that this percentage of images of the real class have been labelled correctly.

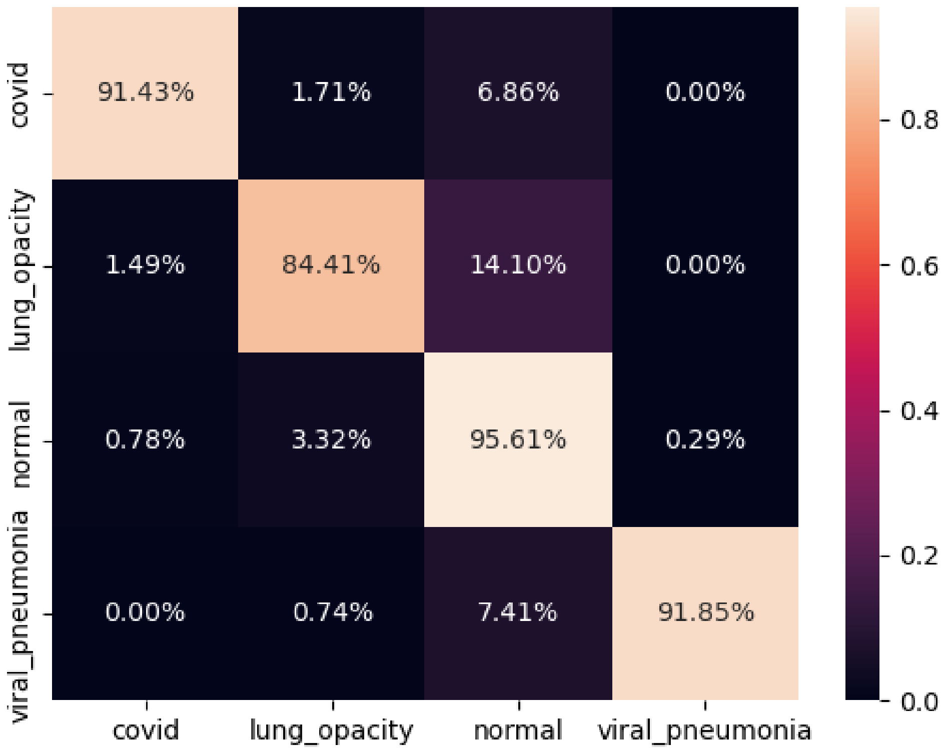


Figure 3.29. Confusion matrix for LMAP3

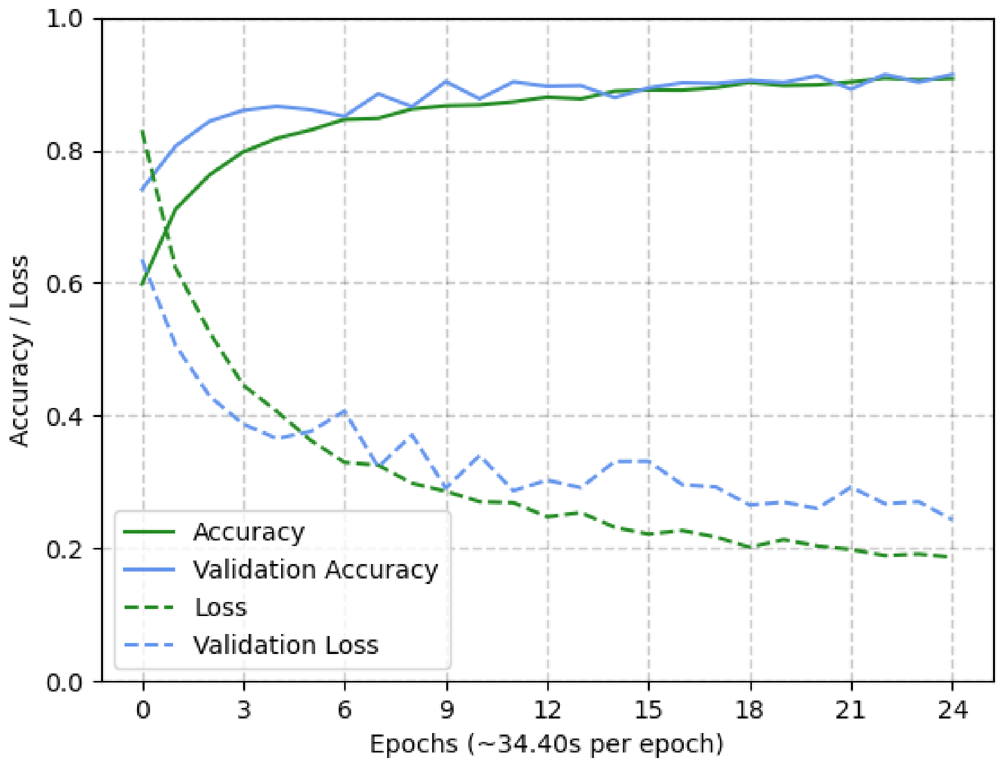


Figure 3.30. Accuracies and Losses as a function of Epochs

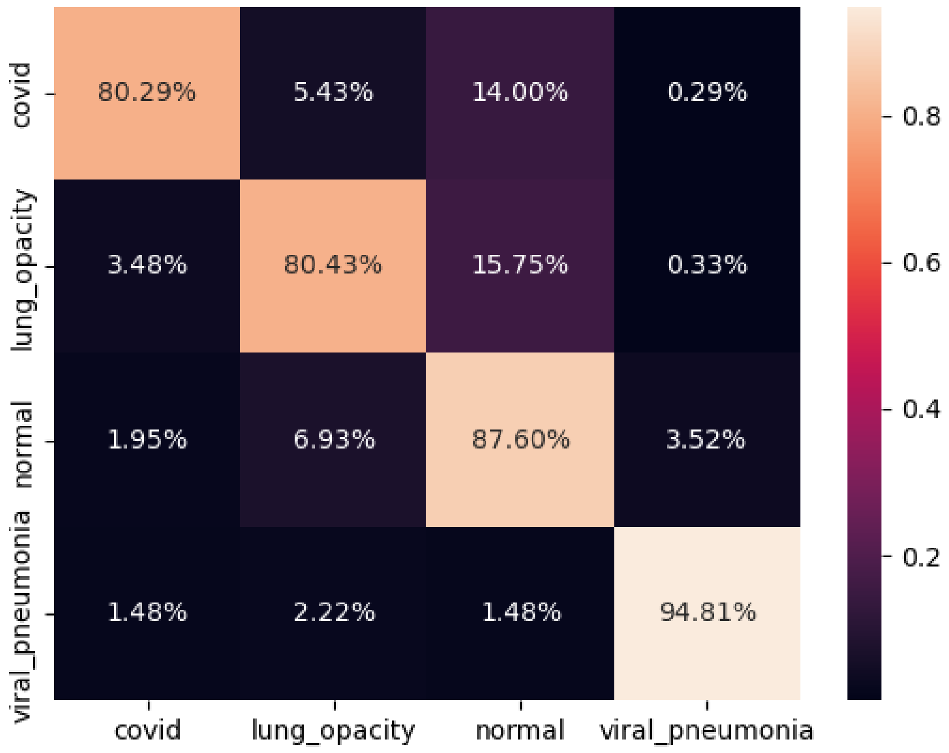


Figure 3.31. Confusion matrix for AlexNet

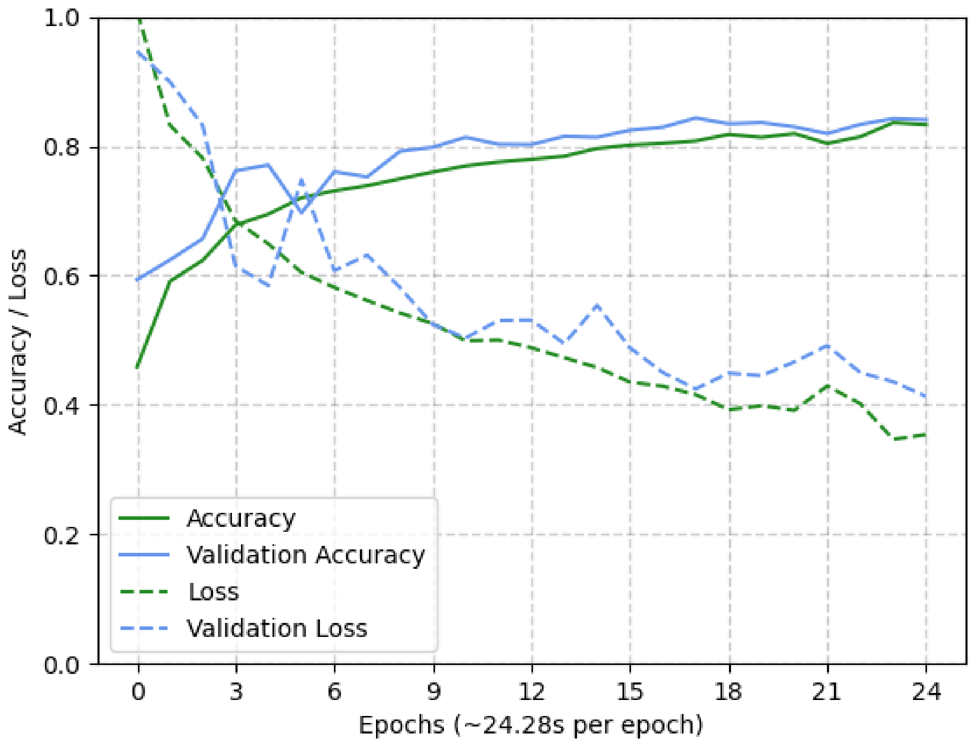
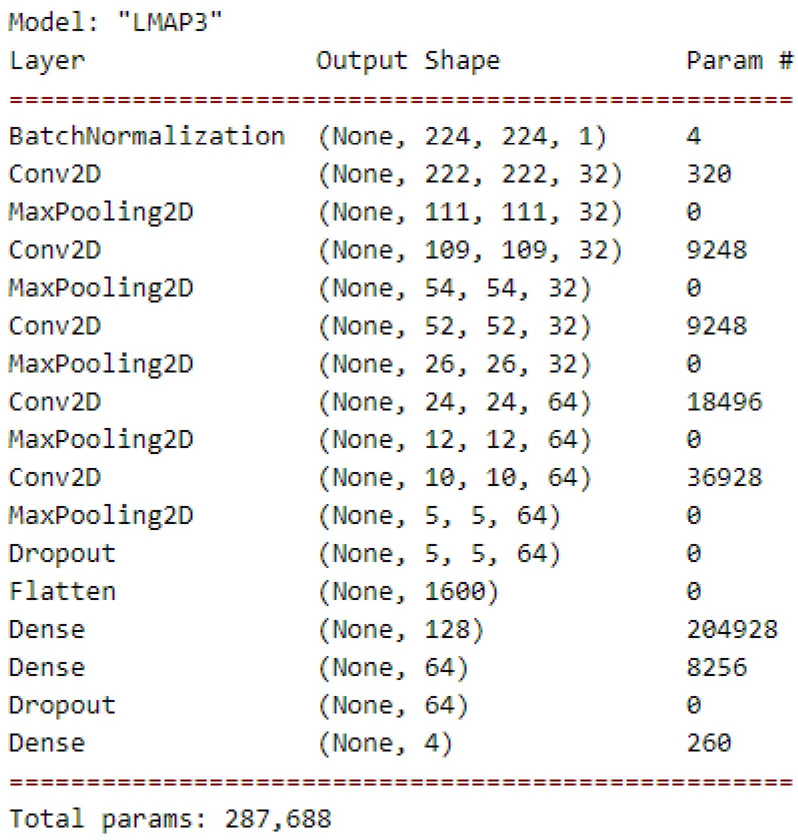
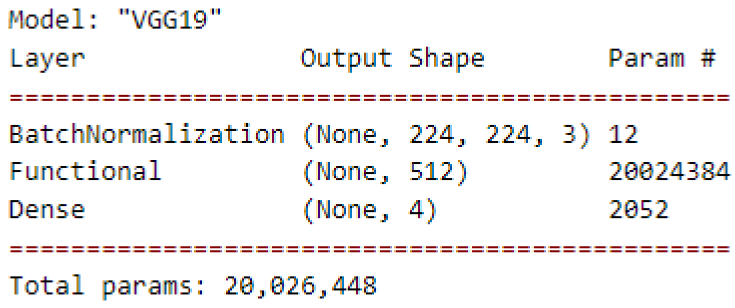


Figure 3.32. Accuracies and Losses as a function of Epochs



```
Model: "LMAP3"
Layer                   Output Shape             Param #
====================================================
BatchNormalization  (None, 224, 224, 1)      4
Conv2D                  (None, 222, 222, 32)     320
MaxPooling2D            (None, 111, 111, 32)     0
Conv2D                  (None, 109, 109, 32)     9248
MaxPooling2D            (None, 54, 54, 32)       0
Conv2D                  (None, 52, 52, 32)       9248
MaxPooling2D            (None, 26, 26, 32)       0
Conv2D                  (None, 24, 24, 64)       18496
MaxPooling2D            (None, 12, 12, 64)       0
Conv2D                  (None, 10, 10, 64)       36928
MaxPooling2D            (None, 5, 5, 64)         0
Dropout                 (None, 5, 5, 64)         0
Flatten                 (None, 1600)             0
Dense                   (None, 128)              204928
Dense                   (None, 64)               8256
Dropout                 (None, 64)               0
Dense                   (None, 4)                260
====================================================
Total params: 287,688
```



```
Model: "VGG19"
Layer                  Output Shape          Param #
================================================
BatchNormalization (None, 224, 224, 3) 12
Functional             (None, 512)           20024384
Dense                  (None, 4)             2052
================================================
Total params: 20,026,448
```

3.3.4. Results

Two of the utilised architectures (LMAP3 and AlexNet) show a good match between the training and validation metrics, which means that they are not overfitting. The goal metric, accuracy, is above 0.8 in all cases, which is rather solid. An accuracy of 0.9 is reached for the LMAP3 model which is close to the best achieved values of ~0.95 found in available solutions on Kaggle.

With regard to the confusion matrix another point that can be observed is that the trained models seem to capture classes differently well. The biggest misclassification comes from images that belong to the category ‘Viral Pneumonia’ and ‘Lung Opacity’ but are labelled as Normal. It could be argued that these two symptoms are less distinguishable than the lungs affected by Covid. Since the majority of our samples are of class 'Normal' it could be that the linear class weight compensation is only efficient to a certain point.

Also it is very interesting to observe that VGG19, a usually highly functional image recognition network, achieves a minor result despite demanding 10 times the amount of computation time. At the same time the trained VGG19 model does present a severe gap between training and validation accuracy, implying that it improved steadily on the training data while not generalising well to unseen data. Overfitting is a common issue and can be addressed with several ways, e.g. data augmentation or the addition of dropout layers. However since both of these are implemented, and VGG19 is a supposedly high performing pretrained model, it is beyond the worth of the effort to investigate why this one performs so badly in our case.

3.3.5. Interpretability

Model interpretability in deep learning for image classification is crucial for understanding and trusting the decisions made by complex neural networks. As deep learning models, particularly Convolutional Neural Networks (CNNs), become more advanced and widely used, it becomes imperative to elucidate how these models arrive at their predictions. Interpretability techniques, such as saliency maps and Grad-CAM (Gradient-weighted Class Activation Mapping), help visualise which parts of an image are most influential in the model's decision-making process. These methods provide insights into the model's inner workings, highlighting important features and patterns that the network has learned. This not only aids in diagnosing and correcting potential biases or errors in the model but also builds confidence among users and stakeholders by demonstrating that the model's decisions are based on relevant and understandable aspects of the input data. For our project we focussed on the GradCAM approach

GradCAM (Gradient-weighted Class Activation Mapping)

Grad-CAM is a popular technique used for visualising and interpreting the decisions of Convolutional Neural Networks (CNNs), particularly in image classification tasks. It provides a way to produce visual explanations for predictions made by a CNN, highlighting the regions of the input image that are most relevant to the predicted class.

How Grad-CAM Works

1. **Select a Convolutional Layer**

   Typically, this is a layer near the end of the network, where high-level features are captured. The feature map of this layer will be visualised.

2. **Forward Pass**

   Predict the class scores (probabilities) before applying the final softmax function

3. **Compute Gradients**

   Compute the gradient of the score for the target class with respect to the feature maps of the selected convolutional layer. This gradient highlights how changes in the feature maps affect the class score.

4. **Global Average Pooling**

   Apply global average pooling to the gradients obtained in the previous step. This gives the importance weights for each feature map. These weights reflect the significance of each feature map in the decision-making process for the target class.

5. **Weighting Feature Maps**

   Multiply each feature map by its corresponding importance weight. This scales the

   feature maps according to their relevance to the target class.

6. **Generate Heatmap**

   Sum the weighted feature maps across the channel dimension to obtain a single-channel heatmap. This heatmap highlights the important regions in the input image.

7. **ReLU Activation**

   Apply the ReLU (Rectified Linear Unit) activation function to the heatmap to focus on the positive influences. Negative values are set to zero, as they typically represent regions that are less relevant or counter to the predicted class.

8. **Upsample to Original Image Size**
   Upsample the heatmap to the same size as the input image. This can be done using bilinear interpolation or other upsampling methods.
9. **Overlay on Input Image**

   The resulting heatmap is then overlaid on the input image to visualise which parts of the image contributed most to the classification. The regions with higher intensity in the heatmap are more influential in the model's prediction.

To this point the implementation of GradCAM does not yield a satisfying explanation for our model’s ways of working. We hope to implement it by the final report.