ANALYSIS OF COVID-19 CHEST X-RAYS: Report 2: Modeling Report

Saniya Arfin, Yvonne Breitenbach, Alexandru Buzgan

May 2025

Contents

1 Introduction

- What kind of machine learning problem is your project like? (classification of images with 4 classes
- We first tried to solve the problem by using several machine learning methods
- Then tried Deep learning, DNNs, transfer laerning, ...
- What task does your project relate to? (fraud detection, facial recognition, sentiment analysis, etc)?
- ...

2 Machine Learning

2.1 Preprocessing Pipeline for Machine Learning

The following subsections describe the different stages of preprocessing the data has run through before it was used for modelling.

2.1.1 Input

Input: Raw Images (299x299 RGB) A set of raw images, each with dimensions 299x299 pixels, with 3 color channels (RGB) and some L.

2.1.2 Convert to Grayscale

Input: Raw images of size 299x299 RGB (colored images).
Operation: Convert the images from RGB (3 channels) to grayscale (single channel). This reduces the complexity and focus on intensity patterns.
Output: Images of size 299x299, but now with only 1 channel (grayscale). These images are binary-like in the sense that they only have intensity values rather than color.

2.1.3 Apply Gaussian Blur to images

Input: Grayscale images of size 299x299.
Operation: A Gaussian blur is applied to smooth the images and reduce noise. This blurring operation makes the image less sharp but removes high-frequency noise.
Output: Grayscale images of size 299x299, but with smoothed details and reduced noise.

2.1.4 Contrast Enhancement with CLAHE

Input: Grayscale images of size 299x299 to which Gaussian Blur filter has been applied.
Operation: CLAHE (Contrast Limited Adaptive Histogram Equalization) enhances the contrast of the images by redistributing the intensity values, making small details more prominent.
Output: Grayscale images of size 299x299, but with improved contrast.

2.1.5 Normalization

Input: Grayscale images of size 299x299, to which Gaussian Blur and CLAHE filter have been applied, with pixel values ranging from 0 to 255.
Operation: Normalize the pixel values to the range [0, 1] by dividing each pixel value by 255.
Output: Grayscale images of size 299x299, but with pixel values now in the range [0, 1]. The values are floating-point numbers.

2.1.6 Resize the masks

The masks have a size of 256x256 pixels and therefore cannot be combined with the X-ray images which have a size of 299x299 pixels. That is why the masks have been resized to 299x299 pixels.

2.1.7 Convert masks to binary images

The masks have been converted into binary images.

2.1.8 Combine masks with images:

Each of the so far processed images havs been combined with its corresponding mask. This has been done by using the function "bitwise_and" of the cv2-library.

2.1.9 Pixel Value Normalization

ToDo: write text

2.1.10 Label encoding

Input: Class labels like "COVID", "Normal", "Lung Opacity", "Viral Pneumonia".
Operation: Convert categorical class labels into numeric labels using LabelEncoder (e.g., "COVID" = 0,"Lung Opactiy" = 1, "Normal" = 2, "Viral Pneumonia" = 3).
Output: Encoded labels as numeric values, stored in an array (e.g., [0, 1, 0, 2 ,...] for the respective classes).

2.1.11 Split data into training and testing

Input: Normalized images (of size 299x299) and encoded labels.
Operation: Split the data into training and testing sets, 80% for training and 20% for testing. We used the paramter "random_state" of the function "sklearn.model_selection.train_test_split" to make sure that the split is reproducable and we use the same train and test sets for working with the differend models.
As we have an imbalanced number of images in the different classes, we used the parameter "stratify" of the function "sklearn.model_selection.train_test_split". This ensures that the train and test datasets have the same proportion of classes as the original dataset. Table 1 shows the number of images in the test and train dataset per class.
The class imbalance will be fixed in the train dataset later, in section 2.1.12.
Output: Two datasets:

- Training set: Images (normalized) and labels (encoded).
- Testing set: Images (normalized) and labels (encoded)

	Original dataset	Test dataset	Train dataset			
			No. of images	No. of augmentations	Augmented images	No. of images (augmented + orig)
COVID	3616	723	2893	2	5786	8679
Lung Opacity	6012	1203	4809	7	4809	9618
Normal	10192	2038	8154	0	0	8154
Viral Pneumonia	1345	269	1076	1	7532	8608
Total number of images	21165	4233	16932	-	18127	35059

Table 1: Number of images per class in the original dataset, in the test dataset and in the train dataset.

2.1.12 Address Class Imbalance by Data Augmentation

Since the distribution between the classes is not balanced (for example Viral Pneumonia images are far less numerous than the Normal images) we have to address this problem before the training stage. Appropriate strategies can be applied, such as:

- Class weighting during model training
- Oversampling/undersampling
- Data augmentation for minority classes

This ensures the model does not become biased toward the majority classes and maintains fair performance across all categories. For our dataset (which consist of images) data augmentation is the most suited.

Input: Training dataset
Operation: Apply augmentation techniques like rotation and shifting to artificially increase number of images in the minority classes "COVID", "Lung Opacity" and "Viral Pneumonia".
Output: Augmented images, which are variations of the original images, still of size 299x299. These augmented images are then used to train the model more robustly. Table 1 shows the number of images per class before augmentation, the number of augmentations per class which were applied and the final number of images in the train dataset. The final train dataset includes the preprocessed, non-augmented images plus the augmented images.
Figure 1, figure 2 and figure 3 show for each of the classes which needed augmentation the non-augmented image and its augmentations. As the class COVID needed 2 augmentations (see 1) figure 1 shows 2 augmented images.

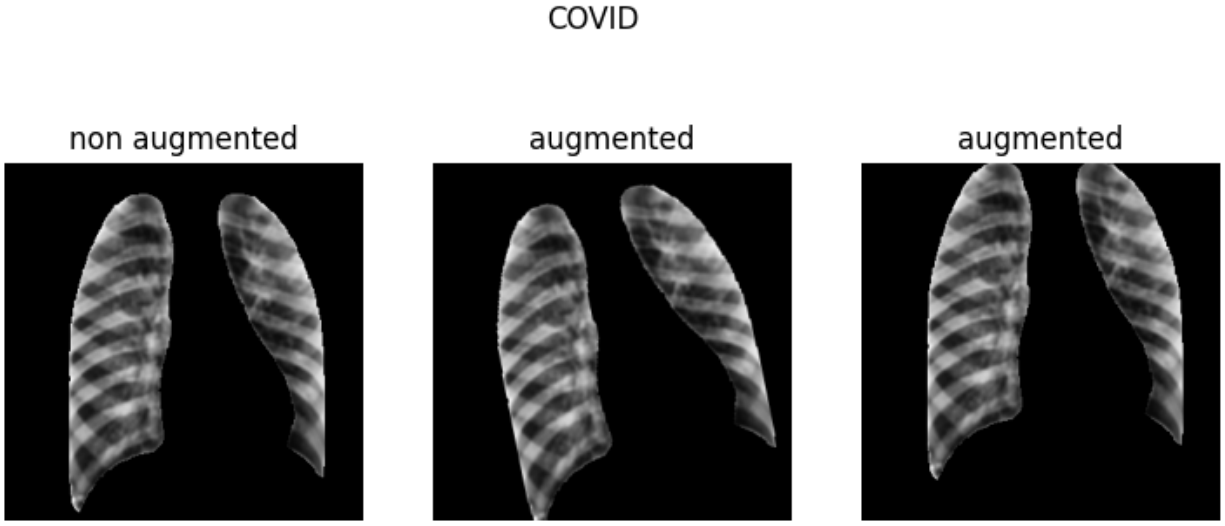


Figure 1: Example of processed X-ray image combined with mask and its augmentations for the class "COVID".

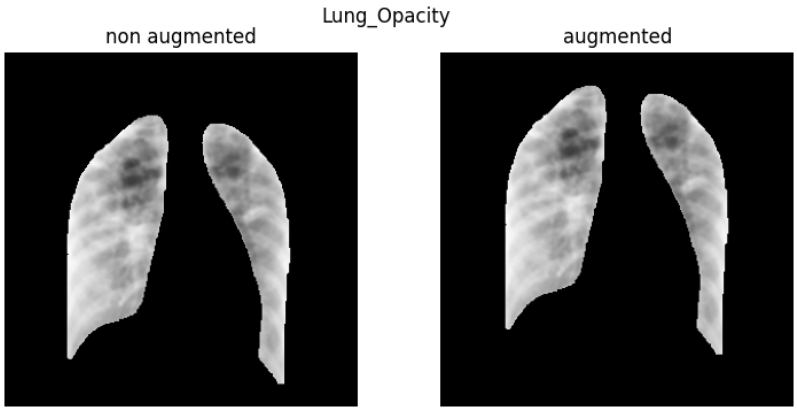


Figure 2: Example of processed X-ray image combined with mask and its augmentations for the class "Lung opacity".

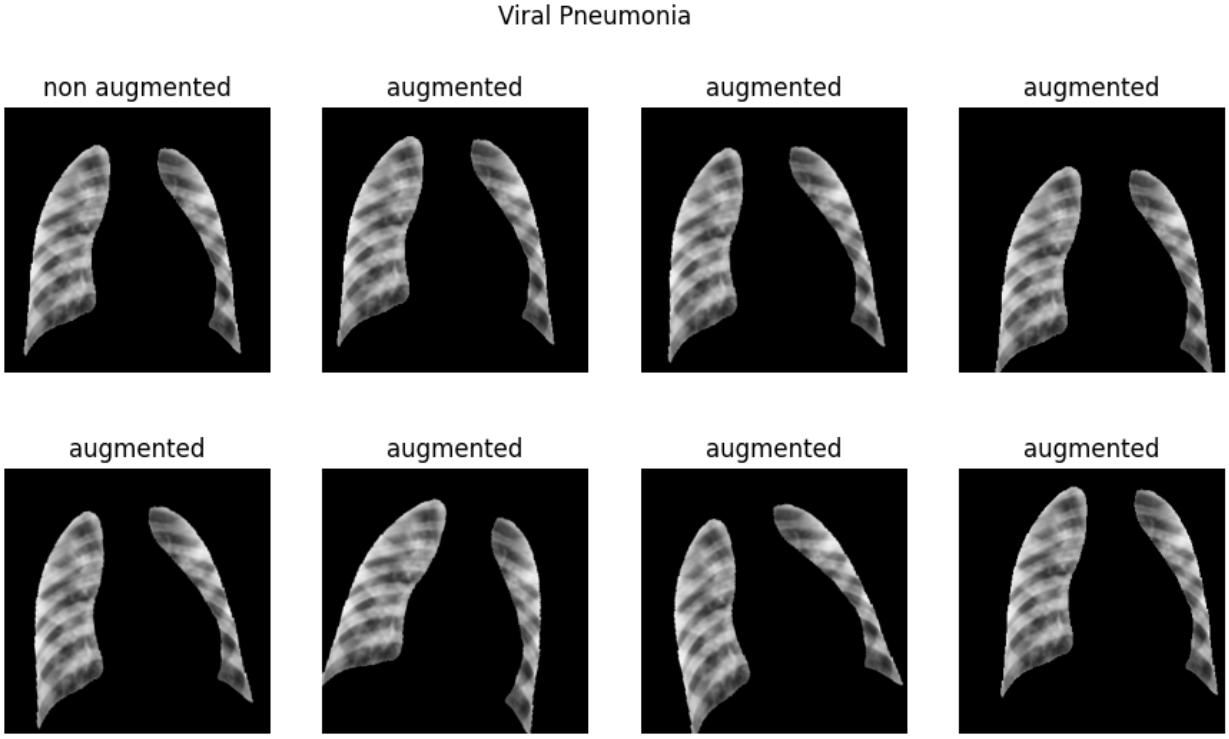


Figure 3: Example of processed X-ray image combined with mask and its augmentationsfor the class "Viral pneumonia".

2.1.13 Feature Extraction

Since we have a large dataset consisting of a total of 35059 images just in the train set it is a good idea to identify important features to reduce the computing time. Extracting features reduces the image's size dramatically and converts it into a compact representation that's suitable for machine learning models.

Chest X-ray images reveal structural differences in the lungs based on the condition. For example:

- COVID-19 often shows bilateral ground-glass opacities.

- Viral / Bacterial Pneumonia can show localized consolidation or patchy opacities.
- Lung Opacity is a broader label that includes a range of conditions affecting lung transparency.
- Normal lungs have well-defined, clear lung fields.

To classify these visually distinct conditions, we need a way to extract structural patterns from grayscale X-ray images.

HOG:

HOG, or Histogram of Oriented Gradients, is a feature descriptor used in computer vision and image processing. HOG captures the structure and edge directions in an image, which are crucial for identifying shapes and patterns.

HOG Captures from X-rays

- Edges of lung boundaries, rib cage, and lesions.
- Orientation and distribution of densities — darker/lighter areas correspond to tissue and opacity differences.
- Shape and spread of abnormalities, like how diffuse or sharp an opacity is.

Although HOG is handcrafted, not learned and so it may miss subtle texture or high-level patterns, it is a fast, interpretable, and memory-efficient way to extract features from chest X-rays.
We have a HOG (Histogram of Oriented Gradients) feature dataset with a shape of (35059, 8100), which means:

- 35059 represents the number of images in your dataset.
- 8100 represents the number of features extracted from each image using the HOG method.

These 8100 features are typically a flattened representation of the gradients and orientations from the original image. Our image was originally a 299x299 pixel image, the HOG algorithm works by breaking the image into smaller cells (e.g., 8x8 pixels per cell), compute the gradient orientations, and then describe the image based on these features. The 8100 features might correspond to a smaller, downscaled version of the original image, where each image is transformed into a set of descriptive gradient-based features, rather than keeping the full raw pixel values. In simpler terms, we are working with feature vectors derived from each image, not the full size of the image. Each of the 8100 features corresponds to a particular characteristic (such as the gradient of pixel intensity) of the image after applying the HOG technique. So, (35059, 8100) indicates that we have 35059 images, and each image has been transformed into a feature vector of length 8100 after HOG processing.

Figure 4a shows the original grayscale image. We can clearly see the chest X-ray with anatomical structures like ribs and lungs.
Figure 4b effectively displays edges and gradients, especially around the rib cage, clavicle, and lung boundaries. The blocky structure is normal due to the cell/block division of the HOG algorithm (e.g., 8x8 pixels per cell). The highlighted edges capture shape and orientation information, which HOG is designed to emphasize. The features look rich and well-distributed and are therefore useful for capturing structural patterns:
Figure 4c displays the distribution of HOG Features per image. Most feature values are near zero (common in HOG due to sparse gradients) in the histogram and a long tail which represents stronger edges or corners.
The line plot shows the average HOG values across feature indices for each class used for class separability analysis (Figure 4d). It helps identify which feature indices (spatial areas or orientations) contribute more for specific classes. Overlapping curves indicate similar structures between some classes (e.g., viral vs lung opacity).

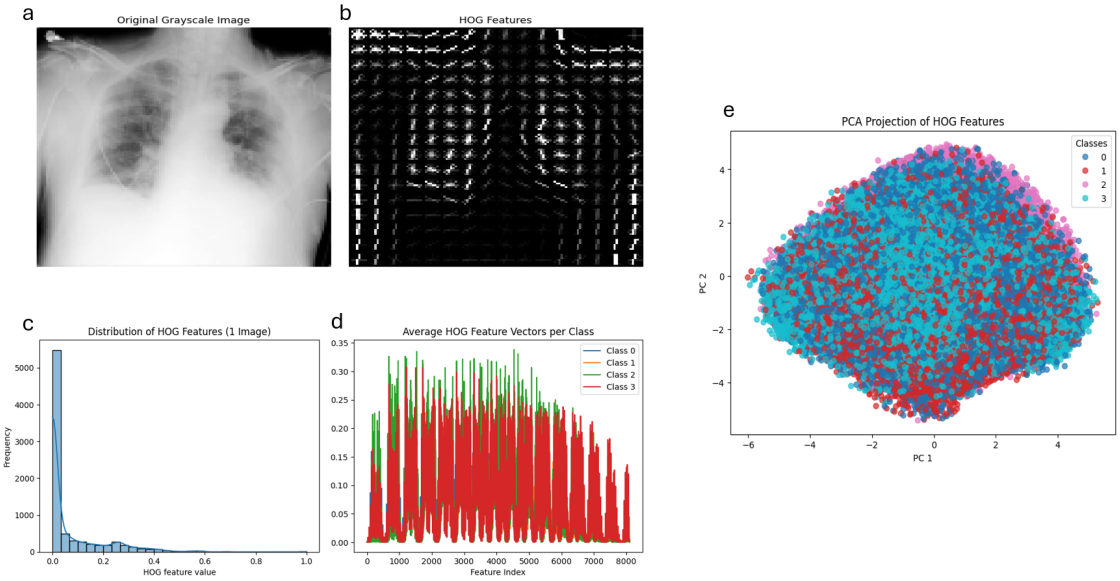


Figure 4: Visualisation of hog features

We also plot a PCA projection of HOG characteristics (figure 4e). The 2D scatter plot of HOG characteristics after reduction of dimensionality using PCA (Principal Component Analysis) showed significant overlap suggesting that HOG alone may not strongly separate some classes (hence the need for a classifier). In summary, HOG works well visually, but it's not sufficient alone for strong class separation in our dataset. Therefore for most our ML models, we use HOG features but we also extracted deeper features using other methods like ResNet, VGG16 and GLCM.

VGG:
infos and results for VGG-method

other:
infos and results for another method we tried

combined methods: infos and results for a combination of methods

I don't understand if the following itemize another subsebsection or does it belong to the subsubsection "feature extraction

- Flatten Features for ML Models
  Input: Extracted HOG features (1D numpy array) for each image.
  Operation: Flatten the 2D image representation into a single 1D array of features.
  Output: A 1D numpy array for each image that can be used as input to machine learning models. This would typically be an array of shape (n_samples, n_features).

- Training Machine Learning Models:
  Input: Feature arrays (flattened HOG features) and encoded labels. Operation: Use machine learning models (e.g.Random Forest, Logistic Regression, XGBoost) to train on the feature vectors. Output: A

trained model that can predict the class (label) of new, unseen images based on their feature vectors.

2.2 Metrics to Evaluate the Model Results

Input: Trained machine learning model and the test set (augmented images, labels).
Operation: Evaluate the model's performance using metrics like accuracy, F1 score, and confusion matrix to understand how well it classifies the different image classes.
Output: Evaluation metrics to show true vs predicted labels.

There are also a number of hyperpramater tuning strategies, which are summarized below:

Method	Recommended For Us?	Why
GridSearchCV	No	Too slow for SVMs (due to image data + multiple classes)
RandomizedSearchCV	Yes	A great starting point. Fast, simple, and well-suited for ML models like SVM, k-NN, RF
BayesSearchCV	Yes	**(Advanced)** Ideal for tuning deep models (CNNs) or complex ML pipelines with many parameters — faster than Grid, smarter than Random

Table 2: Comparison of hyperparameter tuning strategies

Table 2 shows . . . We used the following hyperparameter optimization strategies:
We have to work on this! Write down what we really used @Alex @Saniya @Yvonne

- RandomizedSearchCV for machine learning models (SVM, RandomForest)as it balances speed and accuracy well.
- BayesSearchCV: tried it for SVM. For deep learning models, especially CNNs, we can be efficient with GPU/CPU time, and so we will use it.
- We tired to work with GridSearchCV but its too slow on our large dataset and so not.

2.3 Modelling and Results

The **COVID X-ray Classification Project Workflow** outlines the complete pipeline for preparing and modeling chest X-ray data to classify COVID-19 and other conditions using machine learning (ML) techniques.

The process begins with loading and preprocessing the raw X-ray images, where they are converted to grayscale, resized, and normalized using the `gray_image.ipynb` script.

Next, labels are encoded into a machine-readable format using `encode_labels.ipynb`, and the dataset is split into training and testing sets with `train_test_split.ipynb`.

At this stage, the workflow branches into three sets.

SET 1 involves using feature extraction methods such as Histogram of Oriented Gradients (HOG) . The extracted features and labels are saved for model training and hyperparameter tuning.

SET 2 applies masks and filters during data augmentation. These images are resized to 128*128 and 20*20 pixels. The augmented images and labels are saved as `.npz` files for use in ML pipelines.

SET 3 preprocesses images without applying filters by resizing and normalizing them, followed by augmentation without masks. The resulting data is then flattened and stored as `.npz` files for traditional ML models.

SET 4 preprocesses images after applying filters by resizing and normalizing them.

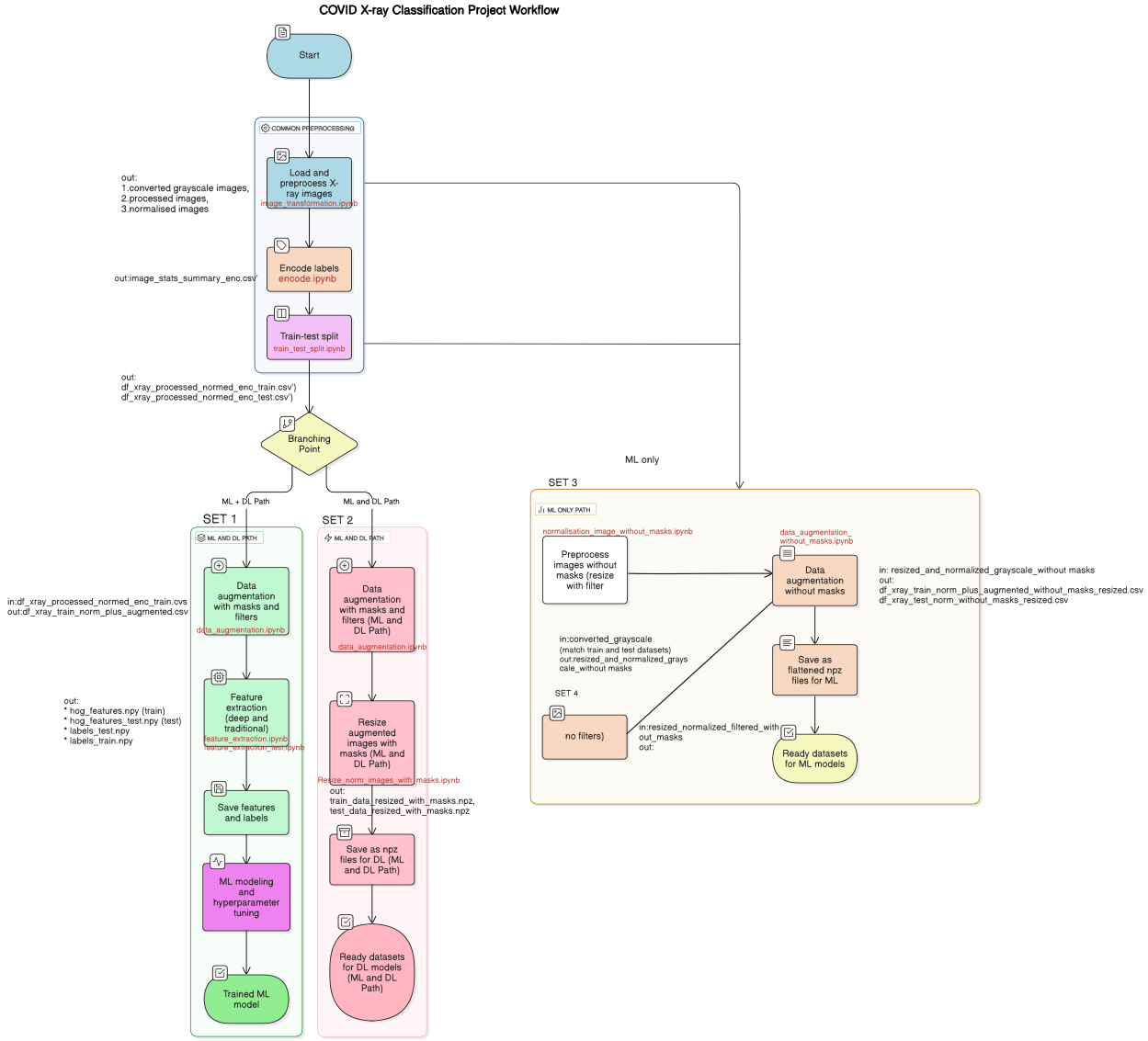


Figure 5: Sets of images used for ML models

With this workflow we wanted to ensure a structured and modular approach to data handling, augmentation, feature extraction, and model training.

2.3.1 Logistic Regression

We started with logistic regression as it serves as a baseline for performance.Also it is computationally inexpensive and we can quickly iterate over different versions, features, and preprocessing methods.In logistic regression, C is the regularization strength while the solver defines the optimization algorithm. Regularization helps prevent overfitting (when our model memorizes training data instead of learning patterns). C is inversely related to regularization:

- A smaller C means stronger regularization (simpler model).
- A larger C means weaker regularization (more flexible model).

Also we need to test different solvers to see which performs best:

- liblinear for binary, small-to-medium-sized data.
- lbfgs or saga for larger datasets or multi-class problems.

1st trial: Hog features(Figure 6A).We gave the model HOG features, a fingerprint of edges. This helped the computer guess pretty well. This model gave an Accuracy of 73.71% Overall, best performance, especially for COVID & Pneumonia.
2nd trial: Images with masks resized(Figure 6B)We shrunk the image small and used masks. It didn't help much. We got an Accuracy: 58.04%. The model made bad guesses, especially for Normal and COVID images however Pneumonia class had high recall (0.90) but low precision (0.42).
3rd trial: Images without masks resized without filters(Figure 6C)we also fed the model same small images, but now the whole image is visible. This increased the accuracy to 69.24% and displayed a nice balance across all disease types.

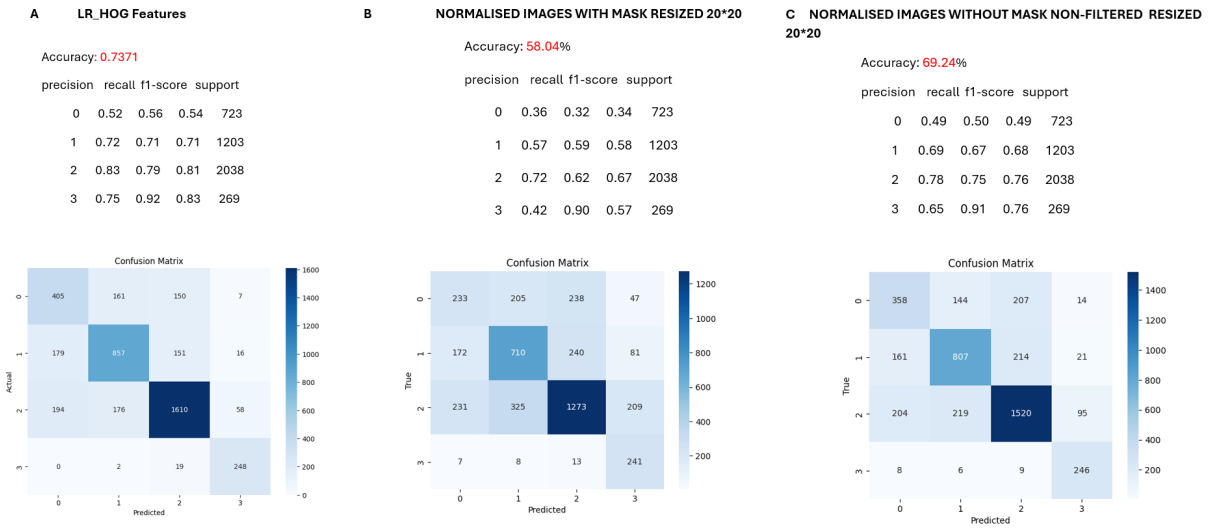


Figure 6: Logistic Regression models A, B,C

4th trial:Random grid search + PCA (Figure 7 A) While running a GridSearch on HOG features, my system ran out of memory and crashed. To make the training more manageable, I switched to RandomizedSearchCV combined with PCA to reduce dimensionality. Here we used SAG(Stochastic Average Gradient)which is ideal for large datasets and when we want faster convergence of SGD without noisy updates. It works by iterating

through the data in mini-batches. In each iteration,it computes the gradient (slope of the cost function) for each mini-batch, and averages them to update the parameters.Over time, this leads to more stable and faster convergence compared to regular SGD(Stochastic Gradient Descent).Also, for dimensionality reduction PCA was used which reduced features to 100 components. Using a random grid with PCA we improved the accuracy to 65.37%, which is solid for logistic regression with HOG characteristics. The best parameters found were C = 0.075, penalty = L1. The PCA retained variance 62% and showed a better precision-recall balance in COVID class.
Class-wise performance:

- Class 2 (2038 samples): Highest F1 (0.74), meaning our model is best at detecting this.
- Class 3 (269 samples): Very high recall (0.93) but low precision — it's over-predicting this class.
- Class 0 and 1: More balanced, but weaker F1. The accuracy drop may be because of the pca which we set to 100.

5th trials: Gridsearch cv (Figure 7B) We applied Grid Search on normalized, unmasked, non-filtered 20×20 images to obtain the best Params: C=0.1, penalty=L2 and an accuracy of 69.10%. Logistic regression after hyperparameter tuning showed slight improvement in accuracy and confusion matrix and displayed a similar class-wise performance to the base version.The raw HOG features might have contained valuable info that PCA compressed away in the previous model.
6th trial:Manual Parameter Tuning using CuML on HOG features (Figure 7C). We then revisited GridSearch with cuML on hog features, which is faster because it uses the GPU. Here we set solver='qn' in cuML's Logistic Regression (i.e., GPU-accelerated version),and it refers to the Quasi-Newton method. Quasi-Newton is a family of optimization algorithms used to minimize a function (like the logistic loss). It is similar in spirit to lbfgs in scikit-learn (also a Quasi-Newton method).It does not support L1 regularization — only L2. In this experiment, we trained a CuML Logistic Regression model using a subset of the training data (5000 samples) with GPU acceleration to enhance performance. The model utilized `class_weight='balanced'` to address class imbalances. This model was fast and efficient and gave the same result as best one (HOG) with an accuracy: 73.47% on the test set.

The classification report reveals that the model's recall is highest for the Normal (2) class (85%)and Viral Pneumonia (3) (82%), while the lowest recall is for COVID (0) (39%). The precision and F1-scores also reflect this imbalance, with the model performing better on the more frequent classes.
Summary:

- HOG features yielded the best accuracy (73.7%) with strong recall on Pneumonia and COVID.
- Raw pixel features (even when resized) underperformed compared to HOG or PCA-based features.
- PCA compressed HOG features to retain 62% variance, resulting in slightly reduced performance but possibly lower model complexity.
- Logistic regression benefited modestly from hyperparameter tuning (Grid and Random Search), but improvements were incremental.
- Pneumonia class consistently showed high recall but variable precision.

In conclusion , the Logistic Regression Model on hog features with Best C: 0.01 obtained by performing manual tuning using Cuml gave us the best accuracy (73%) Room for Improvement: Logistic regression might not be the best choice for such a high-dimensional feature set. Therefore we next chose to experiment with other models like Support Vector Machines (SVM) and Random Forests, which could perform better without requiring dimensionality reduction.

2.3.2 Random Forst

ToDO Alex

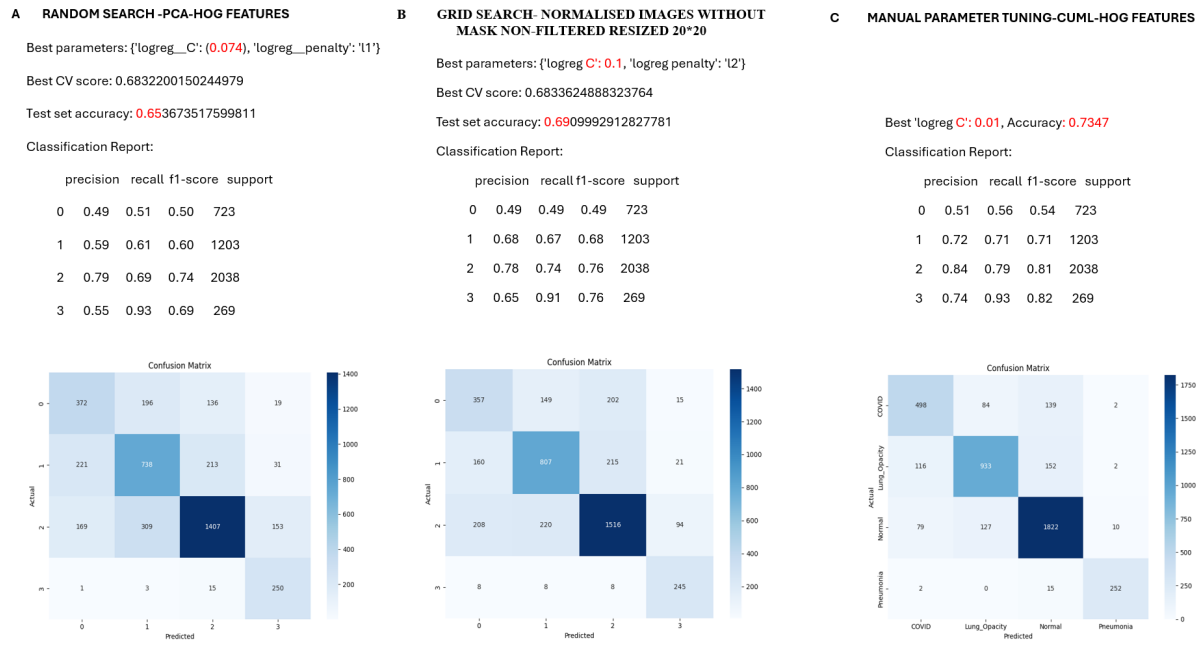


Figure 7: More Logistic Regression models A, B,C

2.3.3 Support Vector Machines (SVM)

There are two types of SVM models: linear and nonlinear SVM. Here's a brief summary of both:
Linear SVM:

- Separates data using a straight line (in 2D) or a hyperplane (in higher dimensions).
- Assumes that the data is linearly separable (i.e., classes can be separated by a straight boundary).
- Uses a linear kernel.
- Fast and efficient for high-dimensional, sparse data like text or HOG features.
- Good for: When the data has clear linear boundaries.

Nonlinear SVM:

- Can separate data with curved or complex boundaries.
- Uses kernel tricks (e.g., RBF, polynomial) to map the data into higher dimensions where it may become linearly separable.
- More flexible but computationally expensive, especially on large datasets.
- Good for: Data where classes are intertwined and not separable by a straight line.

In the following tests we used nonlinear and linear SVMs. As our task is a classification problem we used SVC models.

1th trial
The first trial to run a SVG model was with a dataset with preprocessed images and added masks. The image size was reduced to 128x128 pixels and we tried to use the entire training dataset (approx. 35'000 images). For this first attempt no hyperparameter optimization technique was used. Instead we tried some random hyperparameters:

- gamma=0.01
- kernel='poly'
- other hyperparameter were left to default

After over 6 hours the training of the model was not finished. We stopped it, because this approach had proven as impractical. As it took that much time even without using a hyperparameter optimization technique like GridSearchCV, we needed to change the dataset (number or/ and size of images) for further tests.

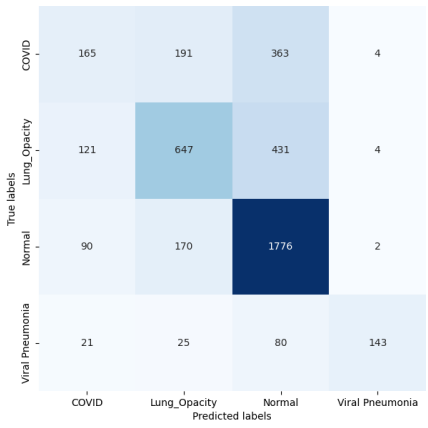
2nd trial
For the second trial we tried to work with a considerably smaller dataset to reduce the training speed. The aim was to get a SVC model running. And maybe in future tests reenlarge the dataset again. Therefore we reduced the size of the images and aswell the number (20x20 pixels and only 2000 images randomly chosen from the train dataset). In this run we tried to use GridSearchCV to find the best parameters out of these options:

- C: 0.1, 1, 10, 100
- loss: 'hinge', 'squared_hinge'
- max_iter: 100, 1000

The best hyperparameter set was: C = 10, gamma = 0.001 and kernel = rbf. The runtime was approx. 12 minutes. The classification report in figure 8a shows that the model has an overal accuracy of 65%, but the prediction of class 0 (COVID) is very poor with an f1-score of only 0.29. The "normal" class could be best predicted which can also be seen on the confusion matrix in figure 8b.

	precision	recall	f1-score	support
0	0.42	0.23	0.29	723
1	0.63	0.54	0.58	1203
2	0.67	0.87	0.76	2038
3	0.93	0.53	0.68	269
accuracy			0.65	4233
macro avg	0.66	0.54	0.58	4233
weighted avg	0.63	0.65	0.62	4233

(a)



(b)

Figure 8: SVC model with 2000 images of size 20x20 pixels, best model with GridSearchCV (C = 10, gamma = 0.001 and kernel = rbf)

3rd trial
Before trying to take more images with higher resulution, we tried to find options to make the training of the model faster.
One idea was to use another method for the hyperpramater optimization. We tested "Bayesian Optimization" instead of GridSearchCV as it does not test every combination of parameters, we hope that Bayes Optimization could be a faster option.
The input data for this test was the same as in the previous test. The "Bayesian Optimization" found the same hyperparameters as in the previous test but finished within only approx. 4 minutes. As this hyperpramater optimization method is about 3 times faster than GridSearchCV we will use this method for the next

test with SVC when we'll try to use more images with more pixels.
Before doing further tests with the SVC model we did some research and found out, that SVC is not the best option for modelling with a lot of data. If using more images and images with more pixels, we increase the number of rows AND the number of columns in our data frame. This means we have a lot more input data for our SVC model. The result of the search in the internet was, that using a linear SVC model could be a better option in our case.

4th trial
As a conclusion of the previous tests we tried out a linear SVC model and used "Bayesian Optimization" for the hyperparameter search.
The number of pixel was increased to 128x128 pixels and the number of images was increased as well. We used 4000 non-augmented images (1000 of each class) to which all preprocessing steps from 2.1 apart from data augmentation and feature extraction have been apllied.

The best hyperparameter set was: C = 0.1, loss = hinge, max_iter=277. The runtime was approx. 1 hour. The classification report in figure 9a shows that the model has an overal accuracy of only 57%, but the prediction of class 0 (COVID) is with an f1-score of only 0.44. With an f1-score of 0.83 the "Viral pneumonia" class (3) is predicted better, which can also be seen on the confusion matrix in figure 9b.

The time needed to fit the model was about 1 hour, which is a lot for such poor prediction results. Maybe the model would perform better if more images or more pixels per image would be used. But due to the long training times we considered a training with more images not as an option.
Therefore we decided to do further tests with a SVC linear model not with the images but with extracted features.

	precision	recall	f1-score	support
0	0.45	0.43	0.44	269
1	0.50	0.51	0.51	269
2	0.50	0.46	0.48	269
3	0.79	0.88	0.83	269
accuracy	0.57	0.57	0.57	1076
macro avg	0.56	0.57	0.56	1076
weighted avg	0.56	0.57	0.56	1076

(a)

(b)

Figure 9: SVC model with 4000 images of size 128x128 pixels, best model with Bayes (C = 0.1, loss = hinge, max_iter=277)

5th trial SGDClassifier(Linear SVM) + HOG (With hyperparameter search) Figure 10A. SGD still supports hinge loss, so it behaves like a linear SVM and also Supports L1, L2, ElasticNet regularization.In this model we used Randomized searchCV best estimator pipeline(Standandr Scaler,PCA, SGD classifier).

```
Best Hyperparameters found were:
{'clf__penalty': 'l1', 'clf__learning_rate': 'optimal', 'clf__alpha': 0.001}
```

We observed low overall accuracy (56.89%),however the recall for the COVID class (class 0) was better at

63.07%, and even stronger for Viral Pneumonia (class 3) at 86.62%.
Perhaps Linear models like SGDClassifier may not fully capture complex patterns. Therefore, the next steps would be to :

- try a Tree-based Model eg. XGBoost or LightGBM which handle non-linear relationships well and work directly on HOG features (no need to resize images)
- experiment with non-linear SVM on a subset of our data (e.g., 2–5k images) to test potential gains

6th trial So the next model we tested was grid search on nonlinear SVM using kernel rbf using a smaller dataset (5000) of hog features.(Figure 10B)We performed grid search with reduced parameter space.The Accuracy improved to 72%, and we found the Best parameters: `svc_ C= 1,svc_ gamma = scale`.
7th trial Since SVC RBF kernel gave us better results we decided to run it on hog features after dimension-

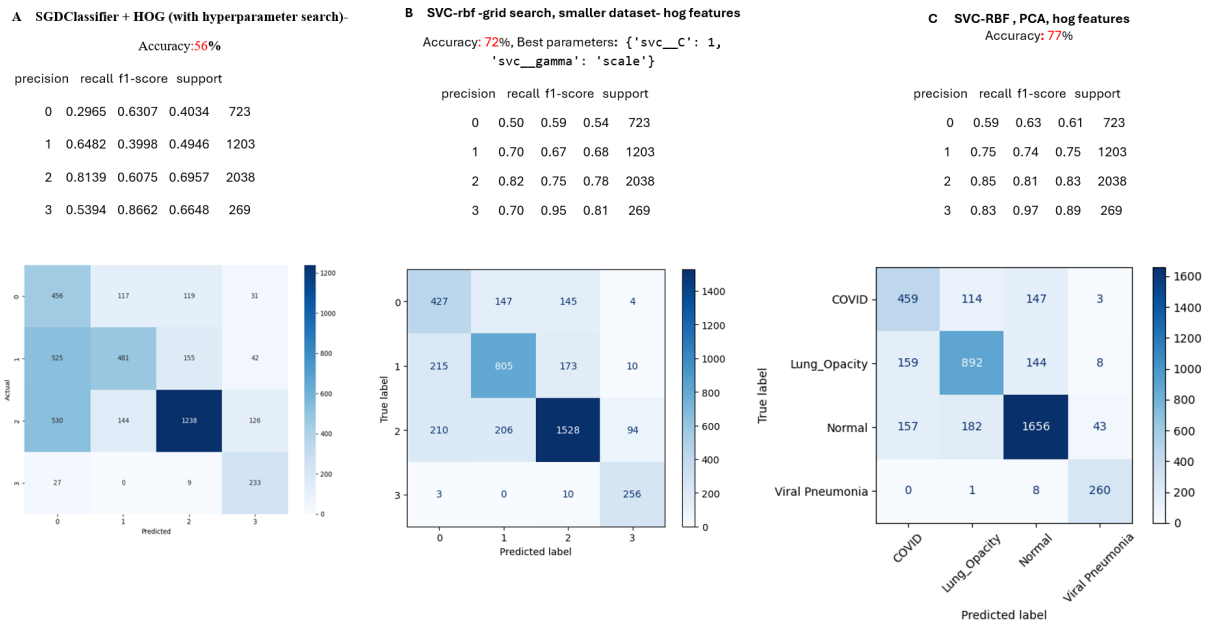


Figure 10: SVM MODELS

ality reduction using PCA. We observed an accuracy of 77%.The Cumulative explained variance for PCA by 200 components was 62.06%. However the precision and recall for class 0 did not show much improvement and therefore, we proceeded with other ML models.

In conclusion, the SVC model with RBF kernel on Hog features with PCA dimensionality reduction gave us the best accuracy of 77%.

2.3.4 XGBoost - Extreme Gradient Boosting

XGBoost (Extreme Gradient Boosting) can be used for classification and regression problems. It is a popular tool often used in machine learning competitions and known for handling large datasets efficiently. After the not very successful tests with e.g. SVM, the use of XGBoost looked promising.

Here is a summary of hyperparameters we used:

- booster = gbtree
- objective = multi:softprob
- learning_rate = 0.3

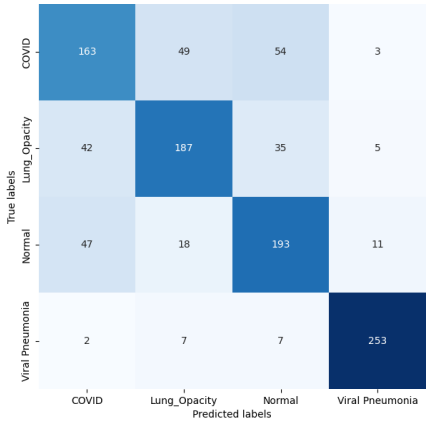
- tree_method = hist
- num_class = 4
- other hyperparameter were left to default

1st trial - XGBoost (128 x 128, non-augmented 4000 images, non augmented, with masks)
For the first test with the XGBoost model we used the same input data as in the 4th trial with the SVM model. The number of pixel was 128x128 pixels and the number of images was 4000 non-augmented images (1000 of each class). To these images all preprocessing steps from 2.1 apart from data augmentation and feature extraction have been apllied.
The training on this dataset took only 5 minutes which was much faster than using an SVClinear model (which took approx. 1 hour).

For this first trial we chose the hyperparameter "number of boost rounds" very high (5000) but additionally used the "Early stopping technique" of XGBoost. Early stopping in XGBoost is a technique used to halt the training process if the model's performance on a validation set does not improve after a specified number of consecutive iterations, thereby preventing overfitting. For our tests we used early_stopping_rounds=50.

	precision	recall	f1-score	support
0	0.64	0.61	0.62	269
1	0.72	0.70	0.71	269
2	0.67	0.72	0.69	269
3	0.93	0.94	0.94	269
accuracy			0.74	1076
macro avg	0.74	0.74	0.74	1076
weighted avg	0.74	0.74	0.74	1076

(a)



(b)

Figure 11: XGBoost with 4000 images of size 128x128 pixels

Figure 11a shows that the overall accuracy was 74%. This accuracy is much higher than 57% which the SVClinear model reached on the same dataset (cf. figure 9). The classification report and aswell the confusion matrix in figure 11b show that the XGBoost model, like the SVClinear model, best predicts the class "Viral Pneumonia" (3), but with a f1-score of 94% instead of only 83%. XGBoost does a better prediction for the class "COVID" (0) with a f1-score of 62% compared to the SVClinear model with only a f1-score of 44%. But a f1-score of 62% is still not very satisfactory. Further tests e.g. with the entire dataset are needed.

2nd trial - XGBoost (128 x 128, all images incl. augmentation, with masks)
As a next test we tried to increase the number of images used for modelling. As input data we used all the images still with a reduced size of 128x128 pixels. To these images all preprocessing steps from 2.1 apart from feature extraction have been apllied.
As the first trial with XGBoost already showed, that we did not need 5000 "number of boost rounds" we reduced it in this test to 500 and additionally kept on using the early stopping technique.
Figure 12 shows that the overall accuracy is nearly the same as in the previous test where we used only 4000 non-augmented images. The f1-score for the "COVID class" (0) is with 57% even a bit lower. But for the other classes the f1-score increased compared to the first test with XGBoost.

	precision	recall	f1-score	support
0	0.55	0.59	0.57	723
1	0.78	0.75	0.77	1203
2	0.84	0.81	0.83	2038
3	0.79	0.96	0.87	269
accuracy			0.77	4233
macro avg	0.74	0.78	0.76	4233
weighted avg	0.77	0.77	0.77	4233

(a)

(b)

Figure 12: XGBoost with all images of size 128x128 pixels, with masks

3rd trial - XGBoost (128 x 128,HOG-features)
For this test we tried to use the HOG-extracted features as input data. The classification report in figure 13a and the confusion matrix in figure 13b show that this did not improve the accuracy of 77% and still the "COVID class" (0) is not predicted very well with an f1-score of only 57%.

	precision	recall	f1-score	support
0	0.58	0.62	0.60	723
1	0.75	0.74	0.75	1203
2	0.85	0.81	0.83	2038
3	0.78	0.96	0.86	269
accuracy			0.77	4233
macro avg	0.74	0.78	0.76	4233
weighted avg	0.77	0.77	0.77	4233

(a)

(b)

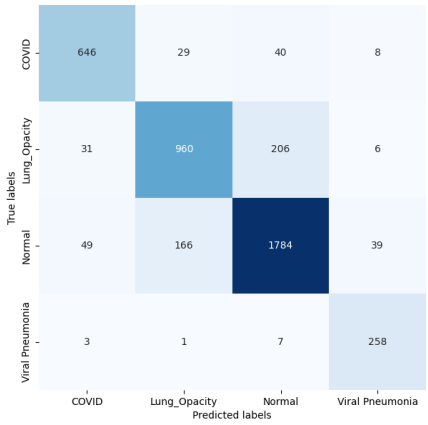
Figure 13: XGBoost using hog features

4th trial - XGBoost (128 x 128, all images, without masks, no filter (Clahe, Gaussian Blur)
As a next step we tried to stick to use a big number of images and used all images incldluing the augmented images as input data. The images have been normalized but no filters and masks were applied.
The classification report in figure 14a and the confusion matrix in figure 14b show that this model has an overal accuracy of 86% which is the best accuracy of all XGBoost models we tried so far. This model can even predict the "COVID class" quiet good with an f1-score of 89%. The worst f1-score has the "Lung-

Opacity class" (1), with still 81There are two options which change in the input data could have made this improvement compared to the previous tests: using no masks or using no filters.

	precision	recall	f1-score	support
0	0.89	0.89	0.89	723
1	0.83	0.80	0.81	1203
2	0.88	0.88	0.88	2038
3	0.83	0.96	0.89	269
accuracy			0.86	4233
macro avg	0.86	0.88	0.87	4233
weighted avg	0.86	0.86	0.86	4233

(a)



(b)

Figure 14: XGBoost with all images of size 128x128 pixels, without masks, no filters

5th trial - XGBoost (128 x 128, all images, without masks, WITH filters (Clahe, Gaussian Blur)
As a conclusion of the previous test (4th trial) we tried to use the same input data but this time using filters. The classification report in figure 15a and the confusion matrix in figure 15b show that using the filters reduced the overal accuracy a bit to 85% and also the f1-score for the "COVID class" (0) is only 84% instead of 89% as in the test without filters.

	precision	recall	f1-score	support
0	0.88	0.79	0.84	723
1	0.83	0.78	0.80	1203
2	0.84	0.90	0.87	2038
3	0.87	0.93	0.90	269
accuracy			0.85	4233
macro avg	0.86	0.85	0.85	4233
weighted avg	0.85	0.85	0.85	4233

(a)

(b)

Figure 15: XGBoost with all images of size 128x128 pixels, without masks, with filters

As a next test we could try using masks and no filters. But as time in this project is reduced we moved on to using deep learning models which looks more promising when working with images.

2.3.5 DENSE NETWORKS: MLP-Classifier

Does this belong to the machine learning? Or is it Deep learning?

But from what we have done with the input data, it suits more to the machine learning part. don't know.

It is somehow inbetween?!!

Multi-Layer Perceptron, which is a classic feedforward neural network consists of: An input layer, One or more hidden layers (each with fully connected / dense neurons) and an output layer.Each neuron in a layer is connected to every neuron in the next layer which makes it a dense network.

1st trial

We trained an MLPClassifier (Multi-layer Perceptron) using HOG features.the first MLP classsifer had the following architecture:sequential 3 dense layers: 512, 256, 128 neurons, ReLU activation + dropout, model compiled using adam optimizer and categorical crossentropy loss.This gave us solid result for a first MLP on HOG features — 78% accuracy with especially strong recall on Viral Pneumonia (97%). In the next trial we improved this further by making improvements to the model architecture.

2nd trial

We increased the number of layers/neurons to make deeper network(Sequential layers: 1024, 512 , 256 and performed BatchNormalization before activations for more stable training, implemented EarlyStopping + ReduceLROnPlateau for smarter training. The overall accuracy of this model was 76.99 %, which is a notable boost from previous models.However, COVID (0) is still the weakest class (59% precision and recall), indicating the model struggles with it — possibly due to class imbalance or feature overlap.Lung Opacity (1) shows consistent, decent performance (75% recall). Normal (2) displayed the strongest precision (85%) and recall (82%), showing good separability.Viral Pneumonia (3) displayed excellent recall (96%) as it had very few false negatives.

3rd trial

The next trial we tried an updated MLP classifier with an edition of Class weights to Boost minority class (COVID) explicitly, and activation function Leaky Relu to avoid dead neurons, in addition to previous additions. The key changes made were:

- LeakyReLU with alpha=0.1 helps mitigate dead neurons by allowing a small gradient when the input is negative.
- Dropout with rates of 50% after the first layer and 30% after the second layer helps prevent overfitting by randomly dropping out neurons during training.
- Adam optimizer with a smaller learning rate (0.0005) to help avoid overshooting the minima during training.
- Class weights are computed to handle class imbalance (especially important for class 0 - COVID).
- Batch Normalization is included after each layer to stabilize the learning process

The accuracy is now 0.7817, which is a noticeable improvement from the previous attempts.Looking at the confusion matrix, we can observe: Class 0 (COVID) has lower precision and recall than the other classes. This could mean our model is having difficulty distinguishing between COVID and other classes (maybe due to class imbalance).Classes 1 (Lung Opacity), 2 (Normal), and 3 (Viral Pneumonia) are being classified well, with high F1-scores and balanced precision and recall.

4th trial

In this model we appliesd SMOTE to help balance the class distribution by generating synthetic samples for the minority classes (Class 0 - COVID), which can help the model learn better representations for those classes.We later scaled the resampled features using 50 epochs, Smaller batch size (32), early stopping, to prevent overfitting, and trained the model on the new balanced data (without class weights). Although the

model gave consistent 77.7–78% accuracy, we observed good precision and recall for class 2 and 3 however class 0 is still underperforming.

5th trial

In this model we tried focal loss which is particularly useful when we're dealing with class imbalance, as it puts more focus on hard-to-classify examples.We used a gamma of 2.0 and alpha of 0.25 — good defaults based on the original focal loss paper. Class 0 is still underperforming slightly (precision = 0.60, recall = 0.58), meaning it's being confused with other classes. This is likely due to it having fewer or noisier samples or its features overlapping with class 1.Class 2 and 3 are performing very well, especially class 3 (recall = 0.93) — so the model is confidently identifying these.

6th trial

We next tried MLP on ResNet features.ResNet (Residual Networks) is a deep convolutional neural network pretrained on ImageNet.It learns hierarchical, high-level features (like textures, shapes, patterns) that are often more effective than handcrafted ones.Extracting features from a pretrained ResNet and feeding them into a simple MLP can boost performance, especially for medical images. The model performed very well with an overall Accuracy of 82.8%, COVID Recall: 69%, Lung Opacity Recall: 78%, Normal Recall: 89%,Pneumonia Recall: 94%.This matrix shows that:Normal and Pneumonia classes are classified with high confidence.COVID and Lung Opacity occasionally get confused with each other and with Normal. Our current results are strong enough that it makes perfect sense to move to deep learning or CNN-based models next. We wanted to see how the MLP classifier model was learning our data anad so we decided to plot the

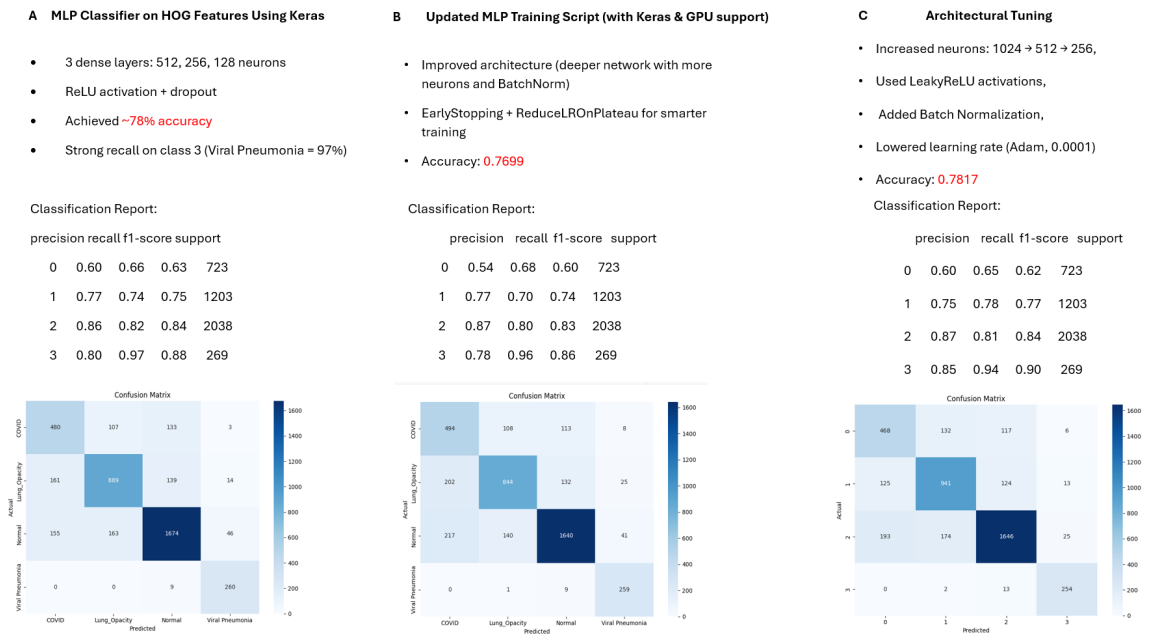


Figure 16: MLP CLASSIFIER A, B, C

accuracy and loss per epochs for the first trial and the last(figure 17). On analysis of figure 17a (20 epochs) we found training Loss decreases steadily suggesting the model is learning from the training data. The Validation Loss Plateaus and then increases after epoch 6–7 which suggests overfitting.The model training Accuracy rises to 0.95 which is excellent however the validation Accuracy flatlines around 0.83–0.84 which showed that the model is not improving on unseen data. Figure 17b (8 epochs) showed a similar trend, just shorter duration. We obeserved still a gap between training and validation performance (train 0.93 vs val

0.83).The validation loss increases even earlier suggesting overfitting happens faster.
These resulsts show that Resnet features wrok well and MLP learns effectively on the training data.However we observe overfitting whih needs to be fixed.

7th trial

To address this we next try to add L2 regularization to Dense layers that helps reduce overfitting by penalizing large weights, increase dropout slightly (0.4) that increases regularization by randomly dropping neurons during training and use early stopping that avoids overfitting by halting training when the model stops improving on validation data.We also try reducing the number of neurons (e.g., 256 → 128 → 64) to avoid overfitting and consider adding BatchNormalization after dense layers. Loss and Accuracy Plot

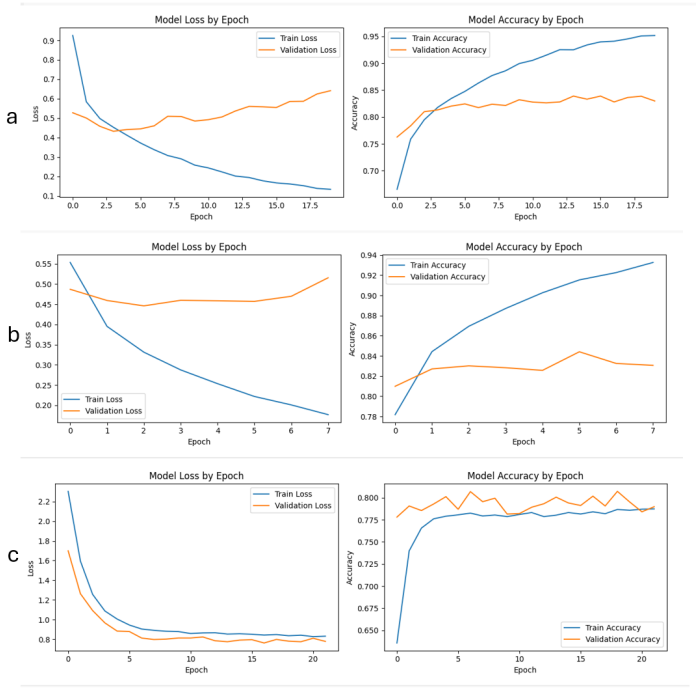


Figure 17: MLP CLASSIFIER A, B, C

Interpretation of figure 17c shows that both training and validation loss drop significantly in the first few epochs and stabilize around epoch 10.The gap between training and validation loss is small, which means low overfitting.Accuracy rises quickly and stabilizes around 78–80% for both training and validation, showing consistent learning and generalization. The validation accuracy occasionally surpasses training accuracy, likely due to dropout and regularization.

The Classification Report showed an overall accuracy of 75% over 4,233 samples which is solid for a multi-class task. Class 2 (largest class): Strong performance (F1 = 0.83), suggesting the model is very confident in this class. Class 1: Also good (F1 = 0.72). Class 0: Weaker (F1 = 0.56), possibly due to overlapping features. Class 3: Interesting — high recall (0.98) but lower precision, suggesting the model predicts this class a lot, possibly overgeneralizing to it.

In conclusion, MLP Classifier on ResNet Features gave us the best accuracy(82%).Since class 0 still remians a probelm moving to a Convolutional Neural Network (CNN) is the logical next step, since were working with image data (X-rays), and traditional ML models like SVMs or MLPs (even with augmentation and tuning) may struggle to capture spatial hierarchies and localized patterns critical for medical imaging.

3 Deep Learning

3.1 Prepare Data for Deep Learning

To Do: As we might have a different preprocessing pipeline we have to describe preprocessing of data we did for deep learning.

3.2 Metrics to Evaluate the Model Results

Do we need this subsection again for DL or is it the same as for ML. Maybe we can delete this subsection later.

Grad-CAM?

3.3 Modelling and Results

3.3.1 Model 1 –> To be replaced with a name

This could be a CNN without transfer learning which Alex is working with.

3.3.2 Model 2 –> To be replaced with a name

This could be a pretrained model.

3.3.3 Model 3 –> To be replaced with a name

This could be a pretrained model.

4 Conclusion

Overall Conclusion Which model performed best What worked, what did not work? Why?

5 Example include image and table - To be removed at the end

Put images you want to include to this report in the subfolder: figures_report_2 and use relative paths inside this report. Upload them on git aswell.

Figure 18 shows ...

X	Y	Z
1	2	3
4	5	
6		7

Table 3: Example of table

Table 3 shows ...

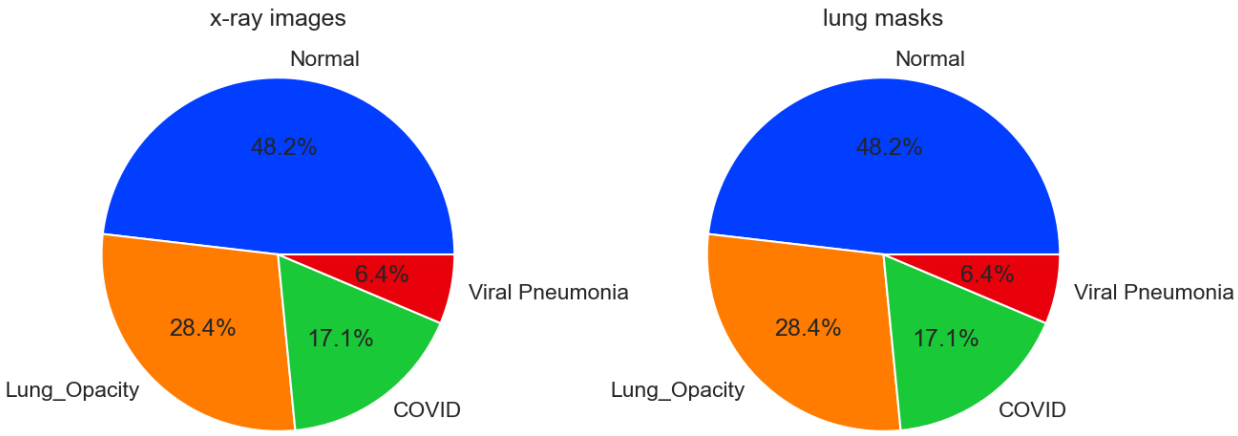


Figure 18: Examples of image