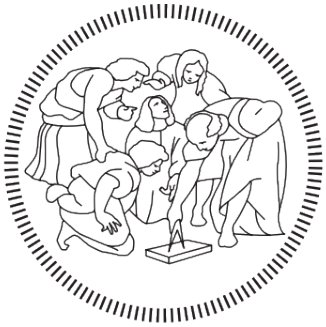


POLITECNICO
MILANO 1863

Pneumonia and Tubercolosis recognition from human lungs X-Ray images

Alice Arneodo, Davide Beretta, Pablo Giaccaglia

January 2023

Contents

1 Introduction

In medicine, X-rays are usually done to view bones and teeth, helping doctors in diagnosing fractures and diseases such as arthritis. A healthcare provider may also request an X-ray to look at organs and structures inside the chest, including the lungs, heart, breasts, and abdomen. Particularly, this paper will focus on the usage of X-ray images of lungs to recognize Tuberculosis and Pneumonia diseases, two of the most significant lung pathologies.

Tuberculosis is a bacterial infection of the Mycobacterium species, namely *Mycobacterium Tuberculosis*. It primarily affects the lungs and spreads from person to person through the air when infected individuals cough or sneeze. Symptoms of Tuberculosis include fever, chills, night sweats, loss of appetite, weight loss, fatigue, and sometimes, significant nail clubbing. When a person has active Tuberculosis, the bacteria are multiplying and causing harm to the lungs and/or other parts of their body, including the lymph nodes, bones, kidney, brain, spine and skin. The progression of the infection can be divided into four stages, occurring over approximately one month: the initial response by macrophages, growth phase, immune system control phase, and lung cavity formation stage.

Pneumonia is a lung infection caused by viruses or bacteria, such as *Streptococcus Pneumoniae*. It inflames the alveoli, the small air sacs in the lungs, causing them to fill with fluid or pus, making it difficult for oxygen to enter the bloodstream. Pneumonia symptoms may present as chest pain during breathing or coughing, confusion or altered mental state, fatigue, a fever accompanied by sweating and shivering, a decrease in body temperature, nausea, vomiting, diarrhea, and shortness of breath.

Being able to identify these diseases is of primary importance for treatments to be effective. However, the examination method depends on the doctor's visual analysis, which can lead to misdiagnosis. Thus, Artificial Intelligence comes to help doctors in this analysis by using deep learning models, such as Convolutional Neural Networks, that can classify the X-ray images given in input. In fact, medical imaging can benefit from deep learning algorithms that can be trained on large datasets to identify patterns and classify images based on the presence of certain medical conditions or diseases. AI models cannot replace doctors, but they are proposed as tools that can help make difficult decisions. As a matter of fact, for what concerns image classification, deep learning algorithms can analyze massive amounts of data to identify peculiarities that a human doctor may miss, leading to more accurate diagnoses. Moreover, these algorithms can significantly speed up the classification process and provide several advantages for clinicians, including increased accuracy and efficiency, reduction of human error and bias, and improvement of patient outcomes. Finally, this can allow doctors to focus more on other tasks such as patient care and consultations.

The goal of this project is to identify the best deep learning models for the purpose of classifying lung X-Ray images into *normal*, *pneumonia* or *tuberculosis*. Moreover, it aims to use *Explainable AI* (XAI) analysis to help clinicians to understand and interpret the predictions made by machine learning models, so that they can make decisions driven by the AI in a clearer way, even when the results are ambiguous.

Part I
Dataset

2 Data exploration

To train, validate and test the models, we were provided a dataset consisting of 15470 X-Ray images in PNG and JPEG format representing both healthy individuals and those with tuberculosis or pneumonia. Each picture is associated with a label, denoting the belonging of the patient to one of the three classes previously introduced. However, the number of images belonging to each class is not balanced, indeed, 60.47% (9354) of the X-Rays are classified as *normal*, 27.47% (4250) as *pneumonia* and only 12.06% (1866) as *tuberculosis*. This can have a severe impact on the performance of the models, therefore it is of extreme importance to try to balance the three sets (training, validation, test) as much as possible. Furthermore, since the images probably come from different sources, the dataset is quite heterogeneous, as it contains images with different dimensions and qualities. The main diversities are now described in more detail.

First of all, since they are the main object of study, the lungs appear in every image. However, it may happen that some pictures contain other parts of the human body, due to the variety of dimensions of the X-Rays. Then, some images are in *negative* format, which means that the bones are black, and the background is white, while in standard radiological format the bones appear white and the background black. Moreover, several samples present different levels of blurriness. Finally, many images present either salt and pepper noise or uniform noise, with different levels of accentuation. In addition to this, there are some images which have been padded in different directions, or present either black spots or superimposed patches.

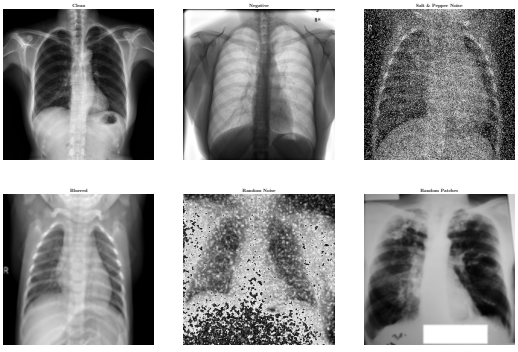


Figure 1: Examples of dataset samples

In order to train the model in an efficient way we need to have a more homogeneous dataset, so an appropriate pre-processing phase is needed.

3 Preprocessing

In order to properly handle the heterogeneity of the dataset, several pre-processing steps were performed before the training phase.

3.1 Pair reduction

We noticed the presence of several pairs of images representing the same acquisition, where one of the 2 samples appears to be substantially affected by image artifacts, so we decided to perform a first preprocessing step which keeps the cleanest of the 2 images for each pair. This step was possible because each of these pairs has a file name belonging to the same patient. To perform the cleaning each pair is processed through a Laplace of Gaussian (LoG) filter to detect noise in an image. The LoG is a second-order derivative filter that enhances the edges of an image while suppressing the noise. It works by smoothing the image using a Gaussian filter and then taking the Laplacian of the smoothed filter. It is worth saying that the Gaussian filter is effective at suppressing additive white Gaussian noise, while it may noy work well at removing other types of noise present in the dataset, such as salt and pepper noise, yet it turned out to be generally effective for the detection task. The Laplacian operator highlights the regions of the image where there are rapid changes in intensity, which correspond to edges and noise. Considering the case of a pair composed by a noisy and a clean image, the Gaussian filter application results in a blurred version of the clean image and in a both blurred and smoothed from noise version of the clean image. The Gaussian output is then processed with the Laplacian operator, which acts as an edge detector on both images, producing an acceptable result for the clean image and a highly disturbed result for the noisy one, since grain impact on the edge detection is predominant. Given the fact that the resulting noisy image presents a bigger amount of high value pixels, the heuristic approach we adopted consists in computing the pixel values' sum for both images, keeping the one with the lowest value.
As a result, the dataset was reduced to 12086 samples.

3.2 Pipeline

The second step of data pre-processing was a pipeline applied to all the images. First of all the pictures have been resized to 224×224 and converted to RGB by stacking 3 times the grayscale channel. This operation was done because some of the proposed models were pre-trained on the *ImageNet ILSVRC2012 dataset*, hence they needed three-layered images in input to be fully exploited. After the conversion, if requested by the training process for a given model, several functions were applied with the attempt to bring all the images to a standard format from the visual point of view. Here's a brief overview of the 4 applied functions. Note that the sequence of explained functions reflects the actual sequence of application.

1. **Negative images detection**: To perform the detection and the inversion of negative samples each image is first binarized in an automatic way through Otsu's thresholding. After doing this, the separation between darker and lighter areas becomes more evident. Since the negative images of the dataset are characterized by a white background easily visible on the 4 corners of the image, 4 corresponding square ROIs are extracted and the overall number of 0 and 1 pixels are considered. If the number of 1 pixels is the majority, the image is detected as negative and thus inverted.

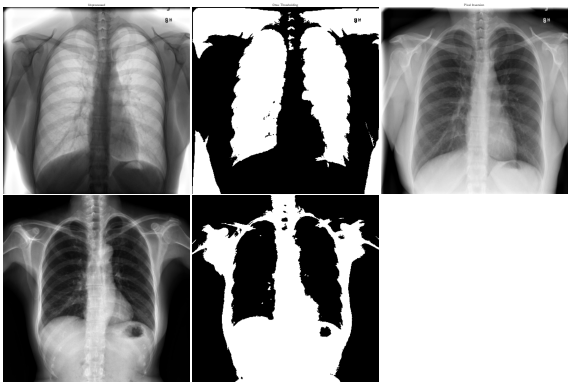


Figure 2: Example of detection process for a negative image and a regular one. After the Otsu Thresholding the negative image's pixel summation is above the threshold, while the normal one isn't, so it isn't further processed

2. **Underexposed images detection**: Since underexposed images are characterized by low valued pixels, first the mean pixel value is computed and, if under a given threshold, the corresponding image is detected as underexposed. Then histogram equalization is performed by means of *Contrast Limited Adaptive Histogram Equalization* (CLAHE), which enhances the contrast of an image with particular attention to regions with low contrast. As a result, this procedure improves the visibility of underexposed images by spreading out the available contrast over a wider range of gray levels.

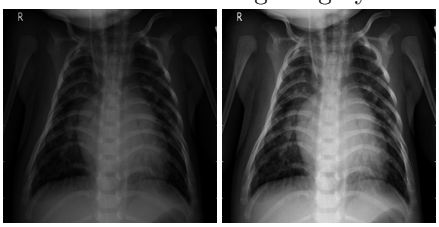


Figure 3: Example of detection process for an underexposed image. In this case the mean pixel value is below the thresholding value of 71, so CLAHE is applied to it

3. **Noisy images detection**: To detect and mitigate noisy images first a LoG filter with a high standard deviation value is applied, to let the blurring have a high impact. The aim of this is to have a clear distinction between clean and noisy images, even in the hardest cases where the grain amount is very low. In fact the result of the Laplacian filter on these highly blurred images is mainly a black pixels image for both cases, with the difference that the noisy ones, even in the hardest cases, present some gray patterns related to the *soft edges* detected after the blurring process. To accentuate the difference between the 2 cases, CLAHE is applied in an extreme way through a very high contrast limit value, which gives high levels of freedom to noise in terms of overall contrast impact, and very high tile size. The result of this

process is a particular processed image which turns out to have a high variance in the pixel values in the case of noisy samples. As a consequence of this if the resulting variance is higher than a given threshold, both *median blur* and *uniform* filters are applied. The rationale behind these is now explained. Median blur is typically used to remove *salt and pepper noise*, which appears as small white and black dots randomly scattered across the image. The median filter replaces a pixel value with the median of the values of surrounding pixels, effectively removing the noisy pixels. On the other hand, a uniform filter is used to remove *Gaussian noise*, which appears as a smooth, randomized variation of the pixel intensities. It replaces a pixel value with the mean of the surrounding pixels, thus an averaging process is performed to reduce the noise.

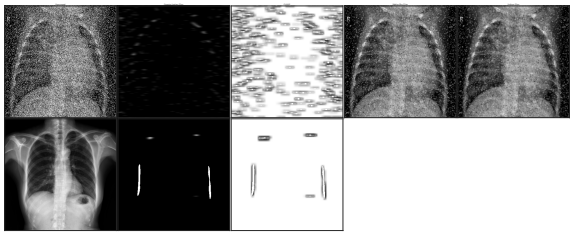


Figure 4: Example of detection process for a noisy image and a regular one. From left to right: unprocessed image, LoG output, CLAHE output, Median Blur output, Uniform Filter output. Notice how after the CLAHE filter on the processed normal image the result is a very low variance output.

4. **Blurred images detection**: It is important to note that while the median and uniform filter are effective in reducing noise, they increase the overall level of blurriness. To overcome this issue blurred images are properly handled, with particular attention to already blurred samples that may have reached an unsustainable blur level after the previous preprocessing steps. As a first step a LoG filter is applied, with the aim of obtaining an image with few detected edges in case of a blurred sample. Then a CLAHE with small contrast limit value and tile size is applied to accentuate the contrast differences. This step is justified by the next one, which consists in binarizing the image according to a minimum threshold of 2. This means that the new pixel value becomes 255 if the original pixel value was above 2, 0 otherwise. This mainly affects blurred images that are turned into almost completely black images. The reason of this is that since the LoG filter on a blurred image hardly detects edges, it produces an output where the majority of the pixels are close to 0. On the other hand not so blurred images may produce a mixed black and white result, so to decide if the sample is a blurred one or not a very low threshold on the overall variance is set, under which the image is sharpened by subtracting to the original sample the negative of the Laplacian filter's output. This accentuates the edges of the image by making the dark side of them even more darker, and the light side even more lighter.

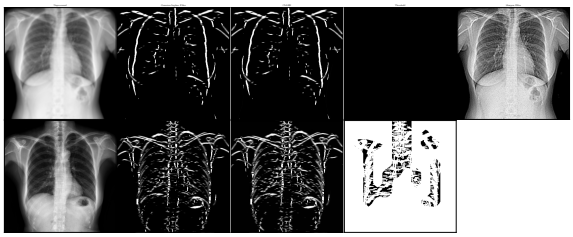


Figure 5: Example of detection process for a blurred image and a regular one. From left to right: unprocessed image, Laplacian of Gaussian filter, CLAHE output, pixel thresholding and Sharpen filter. Notice how after the pixel thresholding on the processed blurred imag the result is a very low variance output

As previously said, out of the four models we implemented, two of them leverage the feature extractor of *EfficientNetV2B3*, and another the feature extractor of *ConvNextTiny*, both pretrained on the *ImageNet ILSVRC2012* dataset. In such cases an additional preprocessing phase is performed, in which the samples are standardized in the range [0,1] and normalized according to ImageNet mean and variance. The remaining one was developed without the need of any pre-trained backbone, so it did not need this pre-processing nor the RGB conversion.
Finally we introduced a small augmentation pipeline aiming to ease the learning process from the generalization and robustness point of view, especially with respect to the undersampled classes.

Part II
Model exploration

In this section a brief overview of the proposed models is presented.

4 ResNet

For our first-shot model we opted to try implementing a deep neural network from scratch and train it specifically for our recognition task, as an alternative to performing transfer learning with neural networks pre-trained on other datasets like Imagenet. We chose to build a residual network because it is of easy implementation and yields much less parameters than other more complex networks, yet it is still quite powerful. Another reason that led us to this choice was the training time: this model required a significant amount of time for the training, more complex networks would have required even longer periods, and given our limited computational resources we couldn't afford it.
The reasoning behind the architecture is provided in the paper "Deep Residual Learning for Image Recognition" [He et al. 2015]. They observed that often, in deep networks, it happens that the accuracy tends to saturate to a certain value, then degradates rapidly. As a result of some experiments, the researchers observed that "Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error" [He et al. 2015]. If we consider

Convolutional Neural Networks, each layer can be seen as the network learning a different task, namely recognizing a higher level pattern in the image (similar reasoning can be made for any Neural Network). They proposed to add shortcut connections which skip one or more layers and add the output of the shallower network to the output of the deeper one. The purpose of these identity mappings is to avoid that the stacked layers produce a training error higher than the shallower network. If this happens " it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers." [He et al. 2015]
The actual architecture of our network is composed of several building blocks that stack an Activation layer, a Conv2D layer and a BatchNormalization, followed by two residual blocks with F(x) = Conv2D, BatchNormalization, Activation, Dropout, Conv2D and BatchNormalization, as shown in the picture.

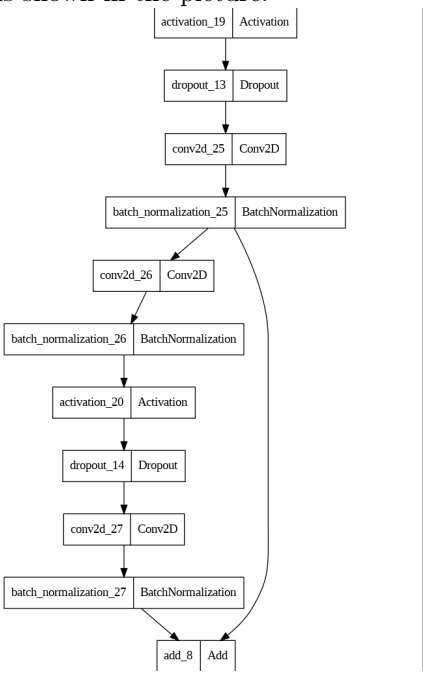


Figure 6: ResNet building block

This, with a couple of layers having 32 filters each at the start, was the convolutional feature extraction part. For classification, we plugged a GlobalAveragePooling layer followed by a Dense layer, a Dropout and a SoftMax classification layer for three classes.
We tried different combination of number of building blocks and number of filters per layer. We started with a small architecture, with just one building block described before and 128 filters per layer. Then we increased the complexity, adding building blocks trying to trade-off training time and the performance of the models. We tried a network with two building blocks, 128 filters per layer in the first block and 256 in the second and an architecture with three building blocks, 64 filters per layer in the first block, 128 in the second and 256 in the third, but none of them achieved good results. The best combination was with three blocks and filters per layer that started from 128 and doubled at each subsequent block, up to 512.
Even in the classification part we tried adding some layers and different number of filters per layer, but the best combination was just one dense layer with 512 filters a dropout layer with 0.1 dropout rate and the output layer with three softmax neurons, one for each class.
This model did not achieve excellent performances, in particular if we compare it with the others we tried, as discussed in the Results section, However we argued that it was worth a shot to try and implement a network from scratch.

5 ConvNextTiny with FFN

This is a *Convolutional Neural Network* (CNN) model based on a transfer learning approach. It consists of a pre-trained feature extraction network, followed by a Global Average Pooling layer for handling the extracted features, and a classification section. This part is composed of a ReLU dense layer with 256 neurons, and a SoftMax dense layer with 3 neurons for the classification task.
The choice of the backbone network was done by testing some *Keras Applications*. After trying various combinations of backbones, such as *ResNet* and InceptionNet, we selected 2 candidate backbones for further experiments: *EfficientNetV2B3* and *ConvNextTiny*. The latter backbone based models will be explained in the following sections. As previously said, we loaded the feature extraction part of the model with the weights learned from the *ImageNet dataset* and experimented with various numbers of layers to freeze during training.
We found the best value for this hyperparameter to be 80 out of 151 backbone layers. This means that we freezed the first 80 layers of the model. The assumption is that the first layers of the backbone were trained on a rich dataset to perform the extraction of general low-level features, thus the weights are already suitable for our classification task and any further training on our data could lead to worse performance. On the other hand, the remaining part of the backbone has a more specialized role in the feature extraction part, so it is more suited to be a starting point for the search for effective weights to extract the most complex features characterizing our dataset. Note that we decided to follow this approach for all the transfer learning models, using this same hyperparameter value, since we found it to be appropriate both in terms of performance and training time.

6 EfficientNetV2B3 with FFN

To tackle the unbalanced dataset and the fact that some classes turned out to be harder to classify than others, as will be shown in the training section, we decided to perform *One-Versus-All* (OVA) classification by building 3 Neural Networks, one for each class. Each of these networks consists of the ImageNet pretrained feature extractor, followed by a Global Average Pooling layer for handling the extracted features, and a classification sec-

tion. This part is composed of a ReLU dense layer with 256 neurons, and a Sigmoid dense layer with 1 neuron for the binary classification task.

7 EfficientNetV2B3 with SVM

The final proposed solution combines both a Deep Learning and a Machine Learning model. The overall model is composed by a feature extraction section which leverages an ImageNet pretrained *EfficientNetV2B3* model, and a classification section that consists of a linear *SVM classifier* which targets the features into different classes. For handling the multi-class problem, the SVM was adopted through the OvA approach. In particular for each class a linear SVM is trained, with the data from the other classes treated as the negative cases.
Denoting the output of the k-th SVM as:

$$\arg\max_x a_K(x) = w^T x \tag{1}$$

The predicted class is $\arg\max_x a_K(x)$. Note that prediction using SVMs is exactly the same as using a softmax. The only difference between softmax and multi-class SVMs is in their objectives parametrized by all of the weight matrices w. Softmax layer minimizes *Cross-Entropy* or maximizes the *Log-likelihood*, while SVMs simply try to find the maximum margin between data points of different classes. The CNN feature extractor and the SVM are jointly trained, allowing a finer network parameters optimization.

8 Explainability

8.1 Why explainability?

In AI-aided decision making, a key role is played by the interpretability of the results and the explainability of the process used to obtain them. Especially in medicine, where a decision could change the life of a patient or even cause their death, it is crucial to build trust in AI tools, both in clinicians, making them consider using them and in patients, so that they know the reasons behind the choices regarding their health. In order to build trust in AI tools one must be able to interpret the result provided by a prediction and somehow explain the process that is behind a particular result even to someone that is not an expert in the field, in our case, clinicians and patients.

8.2 CNNs and explainability

Having this deep convolutional backbones, with tons of parameters, helps a lot with the process of feature extraction, as the convolutional process is able to retrieve from the images information that is useful for the task at hand, without the need of experience in the field. Deep models, indeed, have been introduced to couple the process of classification (or regression) with the feature extraction one, the two crucial steps of machine learning. The main issue with the use of huge architectures is that the results are difficult to examine, interpret and explain, because the features extracted by these networks are often quite different from the ones a person (in the specific case a doctor) would choose. In the following subsections we briefly explain the approaches we adopted for our analysis, which are two popular methods for interpreting the predictions of machine learning models.

8.3 Grad-CAM

For Convolutional Neural Networks, in *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization* [Selvaraju et al. 2019] a method to visualize on the input image which are the regions that have been used for a particular classification was proposed. They called it *Gradient-weighted Class Activation Mapping* (Grad-CAM) because they used the same principle of the CAM algorithm introduced earlier [Zhou et al. 2018], but without modifying the structure of the CNN, as CAM does. The idea is that *convolutional layers naturally retain spatial information which is lost in fully-connected layers, so we can expect the last convolutional layers to have the best compromise between high-level semantics and detailed spatial information*[Selvaraju et al. 2019].
In this sense, Grad-CAM extract the class-discriminative localization map with the steps of algorithm 1.
Given the results in the paper, we argued that Grad-CAM would have been a good way to try to perform some explainability on our convolutional models. So we plugged in a Grad-CAM as described in Keras documentation [fchollet 2020]. With this we were able to visualize the regions of the input images most interested in the prediction of the different classes and we obtained interesting results, as discussed below.

8.4 LIME

Regarding the SVM model we opted for the LIME method to perform XAI. LIME stands for *Local Interpretable Model-Agnostic Explanations* and it is an approach presented in the paper *Why Should I Trust You? Explaining the Predictions of Any Classifier* Ribeiro, Singh, and Guestrin 2016, that tries to help understand the behavior of black-box classifier models, which can either classify tabular data, images or texts. In general, it tries to find the explanation of the black-box model by approximating the local linear behavior of the model providing a solution to understand why the model behaves the way it does. Moreover LIME is model-independent, that is why it is called Model-Agnostic. In short, it generates several samples like the images in input by turning on and off some pixels and makes the prediction of these images using the trained model. Then, it computes the weight of each perturbed image, the more similarity with respect to the original one, the bigger its weight and its importance. Finally, it fits a linear regression model using the weighted perturbed data, so it gets the fitted coefficient of each feature. Hence, the features with larger coefficients are the ones that play a big role in the prediction of the trained model. Scikit-learn al-

lows one to get the probabilities of a SVM prediction by invoking the *predict_proba* method, which is needed to perform the LIME method. However, in the multi-class case it should be done by Platt scaling and Pairwise coupling (as discussed by Wu, C.-J. Lin, and Weng 2004). Hence, we should have trained a logistic regression per class via fold cross-validation, which is very expensive for big models and large datasets. To ease the process, we simply computed the probabilities of each sample by simply using a SoftMax on the prediction scores.

Algorithm 1 Grad-CAM

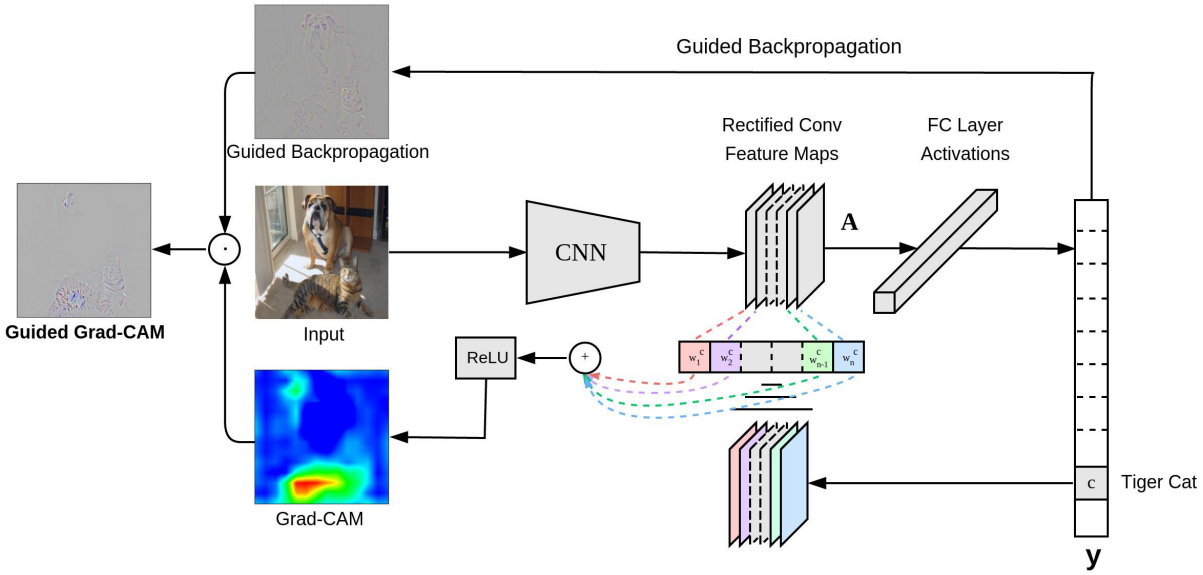
$G^c \leftarrow \frac{\delta y^c}{\delta A^k_{ij}}$ ▷ compute gradient of score for class c

$\alpha^c_k \leftarrow \frac{1}{Z}\sum_i \sum_j G^c$ ▷ compute importance weight through a GAP

$LC \leftarrow \sum_k (\alpha^c_k A^k)$ ▷ Linear combination of importance with fwd activation maps

$L^c_{Grad-CAM} \leftarrow ReLU(LC)$

Figure 6: Grad-CAM architecture



Part III
Results

9 Training

9.1 Loss Functions

9.1.1 Categorical cross-entropy

One of the losses we employed is the Categorical Cross-Entropy, the basic loss function used to train neural networks for classification. It penalizes the probabilities coming out of the SoftMax layer depending on their distance from the expected value, by applying a logarithm. Given a one-hot encoding of the target class, it multiplies it with the outputs of the SoftMax, according to the following equation:

$$CE = -\sum_{i=0}^{C}(t_i log(p_i)) \tag{2}$$

Where p_i is the output of the i-th SoftMax neuron and t_i the value of the one-hot encoding of the target class. We can observe that only one of the t_i, which is at 1, while the others at 0, so we can rewrite the loss as:

$$CE = -(t_c log(p_c)) \tag{3}$$

With c being the index of the actual class.

9.1.2 Focal cross-entropy

This loss function was introduced by T. Lin et al. 2017 in order to handle object detection problems with highly imbalanced classes.

At first, they proposed to weight the cross-entropy loss, multiplying it by a coefficient that measures the frequency of the samples of the specific target class. This weights' task is to give more importance to the least represented classes, in particular:

$$\alpha_i = \frac{N}{\sum_{j=1}^{N} \mathbf{1}(t_j = i)} \tag{4}$$

where N is the total number of samples.

In a second instance they further scale the cross-entropy by a modulating factor $(1 - p_i)^\gamma$, p_i being the output of the i-th SoftMax and γ a tunable parameter.

$$FL(y, p) = (1 - p_y)^\gamma log(p_y) \tag{5}$$

As it is discussed in the paper, if the example is misclassified and p_i is small, then the focal factor grows near 1, while if p_i is closer to 1, the factor shrinks to 0 and the loss for the well-classified sample is reduced. γ increases the effect of the modulation and in particular if it is 0 then the focal loss is equivalent to the Cross-Entropy.

We opted to try and use this loss function because we observed that our dataset was highly imbalanced towards the *Normal* class, as discussed in section 2. We also used the Binary Focal Cross-Entropy loss, that is a variation optimized for binary classification when we trained the networks that discriminate just two classes (OvA models). We set $\gamma = 2$ as it is proposed in the paper[T. Lin et al. 2017], since we could not afford to tune it as well given our scarce computational power.

9.1.3 Categorical hinge

This loss is used for multiclass SVM classifiers. It relies on the traditional Hinge loss, the loss function for binary SVMs. It computes, indeed, the Hinge Loss for taking the prediction for all the computed classes, except for the target class, since loss is always 0 there.

$$L(y) = \sum_{y \neq t} max(0, 1 + w_y x - w_t x) \tag{6}$$

9.1.4 Training Details

To handle the training process of the proposed models a custom Keras Datagenerator was implemented. This choice allowed us to have much more flexibility, since, as it will be discussed in a moment, we didn't opt for an unique training setting, instead each model was trained according to a specific combination of choices, such as preprocessing and augmentation.

The final dataset, obtained after the pair reduction phase, was split into the three sets: Train set, Validation set and Test set. These three sets were used to train all the models and to test them to evaluate their performance on unseen data. Moreover, we performed a hyperparameter-tuning to find the best sets of hyperparameters, such as learning rate and number of layer neurons.

For better handling the Mini-Batch Gradient Descent we employed a Cyclical Learning Rate policy to adjust the learning rate during training by changing it cyclically. This scheduler varies the learning rate between a minimum and a maximum value in a cyclical manner, following in our case a triangular waveform. The rationale behind this choice is that it can help the model to converge faster and avoid getting stuck in local minima during training. In fact a high learning rate can help the model to escape a local minimum and explore new areas, while a lower learning rate can help to refine the weights and stabilize the learning process. We found that a minimum learning rate of 1e-5 and a maximum of 1e-4 were a good compromise between training stability and speed of convergence.

Furthermore, for some models we implemented k-fold cross validation. The reason is that it can avoid overfitting and, since it produces k metrics instead of one, its overall performances are more trustful and data independent.

Finally, Adam optimizer was used and an early stopping callback was adopted to properly handle overfitting, especially with big transfer-learning models.

9.2 ConvNextTiny with FFN

This model was trained with 3 different training settings, now discussed.

9.2.1 Training with no pre-processing

In this first attempt the model was trained without accounting for any pre-processing, apart from the pair reduction, common to all the models here discussed. The training was done using a Categorical Cross Entropy loss. It lasted 22 epochs due to early stopping on validation accuracy. The network reached 100% training accuracy, while validation accuracy reached $97,71\%$. The testing performance was generally satisfactory, with an accuracy of $96,81\%$. Anyway, by a more accurate analysis of the testing performance for each class, we noticed a significant difference in f1-score values between the Tuberculosis class and the others. The results regarding the validation accuracies of the presented models are shown in the Performance Comparison section (11) of this report. In light of this we decided to try to adopt different strategies to tackle this unbalanced performance.

9.2.2 Training with focal loss & class weights

In this second attempt the model was trained by employing class weights and a Sparse Categorical Cross Entropy loss with the aim of forcing the model to focus more on the undersampled Tuberculosis class. The training lasted 10 epochs due to early stopping on validation accuracy. The network reached. 100% training accuracy, while validation accuracy reached $97,29\%$. The testing performance was generally satisfactory, but worse than the one of the previous experiment, with an accuracy of $96,73\%$. The training was much faster but the performance with respect to the various classes was slightly worse than before, with Tuberculosis class still with a significantly lower f1-score. Finally we can notice from the corresponding training plot that the model overfits much more than the previous one, since the gap between the training and validation accuracy is much larger.

9.2.3 Training with data augmentation

In this third attempt the model was trained by employing data augmentation. The included transformations were random shifts, scales, rotations, brightness and contrast alterations and CLAHE. The training was done using Adam optimizer and Categorical Crossentropy loss, focal loss and class weights were not employed to avoid regularizing too much the network. The training lasted 13 epochs due to early stopping on validation accuracy. The network reached 99% training accuracy, while validation accuracy reached $97,71\%$. This training setting differs from the first one only in the data augmentation, which, as can be observed in the training plots' section, helps in avoiding an excessive model over-fitting, since the gap between training and validation loss is significantly decreased, but it leads to worse testing performance, with an even lower f1-score with respect to the previous models.

9.3 EfficientNetV2B3 with FFN

The three models were trained with the same training setting, employing a Binary Focal Crossentropy loss with weight balancing factor α equal to 1 and focusing parameter γ equal to 2, with the aim of forcing the network to focus more on the class of interest for a particular binary classifier, otherwise the learning procedure results to be biased towards the other class, which is larger and easier to classify. In light of the previous experiments, we decided to not employ anymore data augmentation, instead we decided to focus more on refining the learning procedure by including the pre-processing pipeline previously discussed, which was applied to all the three model trainings and to the other model trainings explained later in the report. The training of the Normal class classifier lasted 20 epochs due to early stopping on validation accuracy, the network reached $99,3\%7$ training accuracy and $97,44\%$ validation accuracy. Finally training of the Pneumonia class classifier lasted 9 epochs due to early stopping on validation accuracy, the network reached $99,62\%$ training accuracy and $98,79\%$ validation accuracy. Finally the training of the Tuberculosis class classifier lasted 24 epochs due to early stopping on validation accuracy, the network reached $99,91\%$ training accuracy and $98,08\%$ validation accuracy. Ensemble testing resulted in an increase of overall accuracy, but more importantly an increase of Tuberculosis class f1-score, for which the model achieved $92,09\%$, higher than the previous models, for which the corresponding score didn't go over $90,71\%$.

9.4 EfficientNetV2B3 with SVM

Instead of using an SVM by employing libraries such as Scikit-learn, we decided to implement it in TensorFlow so as to train the entire model (EfficientNet and SVM) through Stochastic Gradient Descent. Notice that the implementation of the TensorFlow based SVM used as a starting point the code made available by by Araya Aftab [Aftab 2022]. To train the SVM model with EfficientNetV2B3 as backbone we used two approaches: first we trained the model using the entire training set and then we performed 5-fold cross validation with majority voting ensembling. Both trainings were performed by using a Categorical Hinge Loss function. Nevertheless, since this model is composed of two consecutive submodels, we could have integrated this loss function with other losses to make the training more refined.

In the former approach, the training lasted 79 epochs, reaching $99,85\%$ of training accuracy and $98,52\%$ of validation accuracy. Also in this case, the testing was acceptable with an accuracy of $97,43\%$. However, due to the unbalanced data set, the performance of the Tuberculosis class was slightly worse than the performance of the other classes, but still good enough.

In the latter approach, we decided to split the train and the validation sets into five folds to perform a 5-fold cross validation. Thus, five different models were trained using 4 folds as the train set and the remaining fold as the validation set. Note that in this way the vali-

dation set is a different fold for each model. The number of epochs of the five models ranged between 37 and 51. Moreover, the average score for the training accuracy of the trained models is $99,85\%$, which is the same as in the first approach, while the average validation accuracy has value $99,61\%$, which is higher by more than 1% with respect to the previous case. Finally, we used the same test set on the five trained methods to perform a majority voting prediction on it. With this method, the test reached an average accuracy of 97.56%, which is very similar to one of the model trained without cross-validation. On the other hand, the general performances moderately increased, maintaining, however, the same difference between Tuberculosis class and the other two classes.

From the training plots in the next section, we can perceive that this model does not overfit, in fact the gap between the training and the validation accuracy is much lower with respect to the other models, and in general it has better performances during the training phase. The loss curves seem to be better too, however we have to take into account that all the other models are using a Cross-Entropy Loss function, or one of its variants, while the SVM model is using a Categorical Hinge Loss function, so they have as objective a different minimization and they cannot be directly compared.

9.5 ResNet

The training of the Residual Network was done with images with just the grayscale channel, because the network did not have a backbone that needed in input RGB pictures. So we removed from the pre-processing the part in which we stacked three times the images on top of themselves and this was the only difference on the pre-processing phase with respect to the other models.

Regarding the loss function, we started by using the Categorical Cross-Entropy, achieving adequate but not too good performances. Later on we tried to address the fact that we had highly imbalanced classes with a Categorical Focal Loss and by introducing class weights to give more importance to the samples of the least represented classes. With these adjustments we obtained slightly better results than training only with Cross-Entropy and without class weights.

As expected, since the network did not have a backbone trained on a huge dataset like ImageNet, it achieved worse results than the other models, as reported in section 11. In particular we observed that this network had a quite unstable learning, with irregular convergence and epochs in which the accuracies dropped abruptly, as shown in graphs 11 and 12. Having observed this, we performed a 5-fold validation and we kept the 5 trained models to perform a majority voting prediction in a Bagging-fashion approach, but it didn't improve the performances, it made them worse instead.

10 Training Plots

Figure 7: ConNextTiny with FNN & no pre-processing

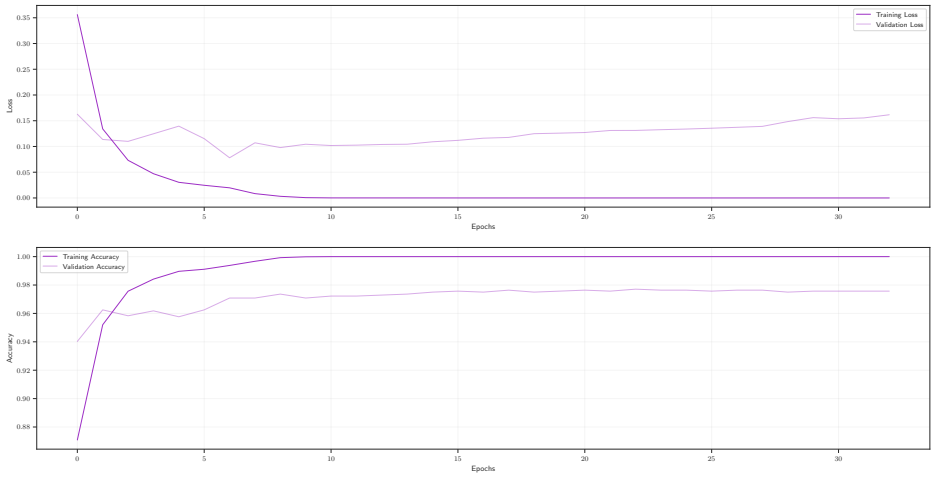


Figure 8: ConNextTiny with FNN & focal loss and class weights

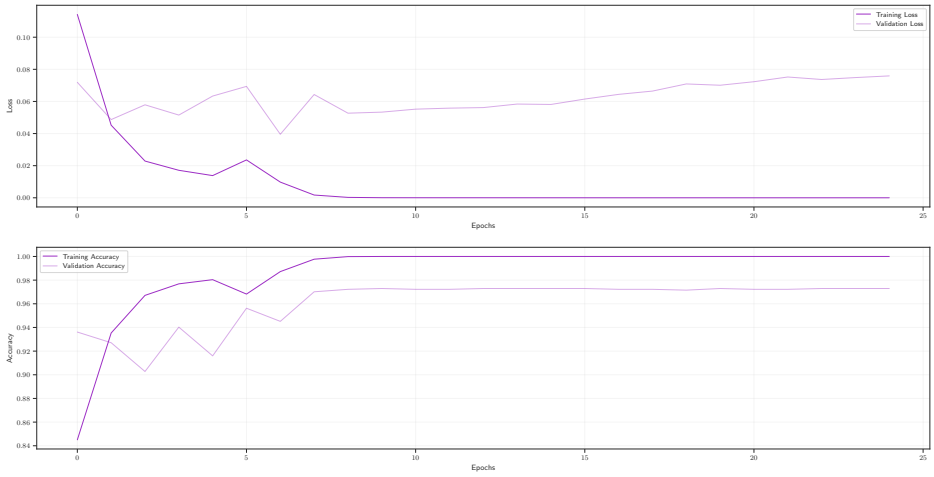


Figure 9: ConNextTiny with FNN & data augmentation

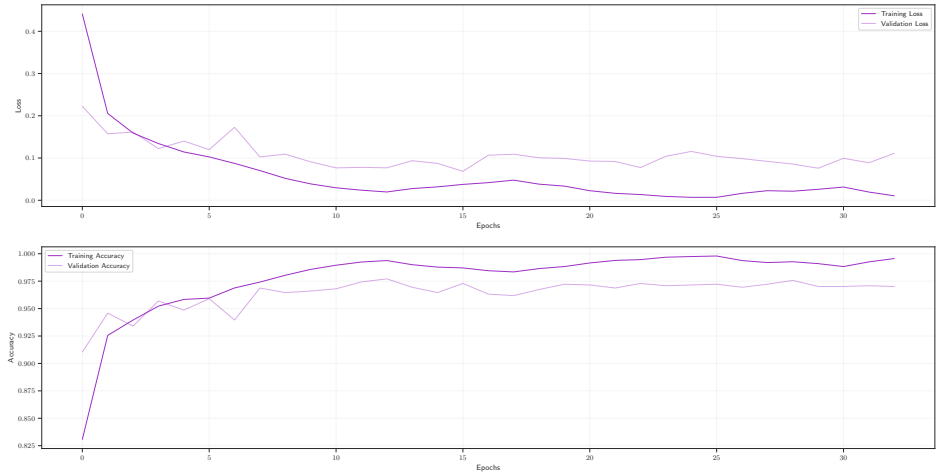


Figure 10: EfficientNetV2B3 with SVM

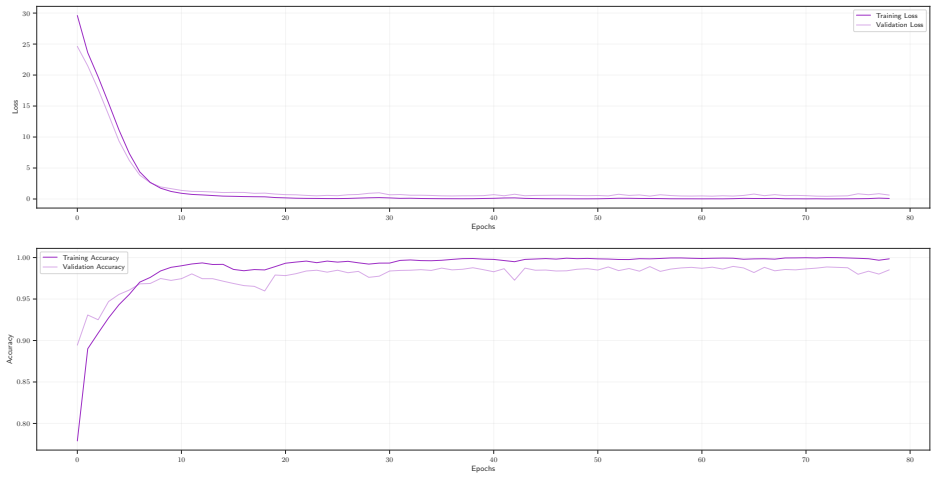


Figure 11: ResNet with Focal Loss and class weights - Loss

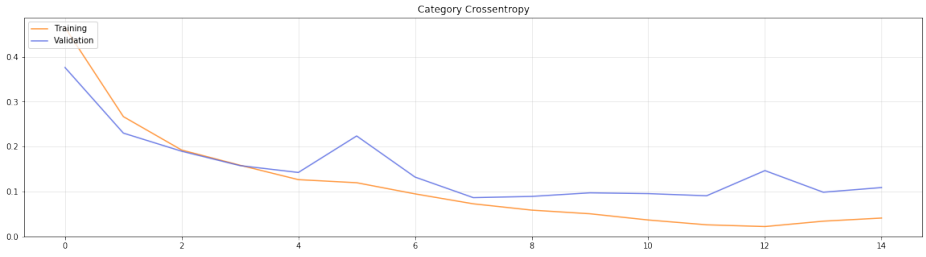
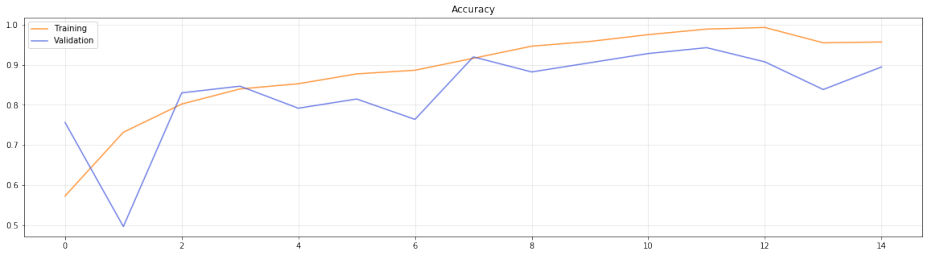


Figure 12: ResNet with Focal Loss and class weights - Accuracy



11 Performance comparison

In the following tables we report the performances over the set of images we dedicated for testing phase of the various models we explored.

Table 1: comparison of overall performance

model	accuracy	precision	recall	f1-score
ConvNext without preprocessing	0.9681	0.9640	0.9452	0.9541
ConvNext with Focal Loss	0.9673	0.9636	0.9414	0.9517
ConvNext with data augmentation	0.9627	0.9538	0.9421	0.9477
EffNet Ensemble	0.9694	0.9712	0.9455	0.9576
CNN SVM	0.9743	0.9725	0.9543	0.9630
CNN SVM Ensemble	**0.9756**	**0.9757**	**0.9577**	**0.9662**
ResNet w/ Cross-Entropy Loss	0.8887	0.8308	0.8949	0.8547
ResNet w/ Focal Loss	0.9412	0.9239	0.9055	0.9141
ResNet Ensemble	0.9367	0.9238	0.8860	0.9028

Table 2: Comparison of performance over the classes

	Normal			*Pneumonia*			*Tubercolosis*		
model	precision	recall	f1-score	precision	recall	f1-score	precision	recall	f1-score
ConvNext w/o prepro.	0.9682	0.9795	0.9738	0.9760	0.9864	0.9812	0.9478	0.8699	0.9071
ConvNext w/ Focal Loss	0.9663	0.9802	0.9732	0.9775	**0.9879**	0.9827	0.9470	0.8562	0.8993
ConvNext w/ data aug.	0.9661	0.9727	0.9693	0.9715	0.9803	0.9759	0.9239	0.8733	0.8979
EffNet Ensemble	0.9652	0.9856	0.9753	0.9787	0.9743	0.9765	0.9697	0.8767	0.9209
CNN SVM	0.9705	**0.9884**	0.9793	**0.9908**	0.9773	**0.9840**	0.9562	0.8973	0.9258
CNN SVM Ensemble	**0.9730**	0.9870	**0.9800**	0.9833	0.9818	0.9826	**0.9706**	**0.9041**	**0.9362**
ResNet CE	0.9559	0.8599	0.9054	0.9287	0.9652	0.9466	0.6077	0.8596	0.7121
ResNet Focal	0.9461	0.9590	0.9525	0.9610	0.9697	0.9654	0.8647	0.7877	0.8244
ResNet Ensemble	0.9339	0.9665	0.9499	0.9680	0.9621	0.9651	0.8694	0.7295	0.7933

12 X-AI

12.1 ResNet

Figure 13: ResNet Grad-CAM results

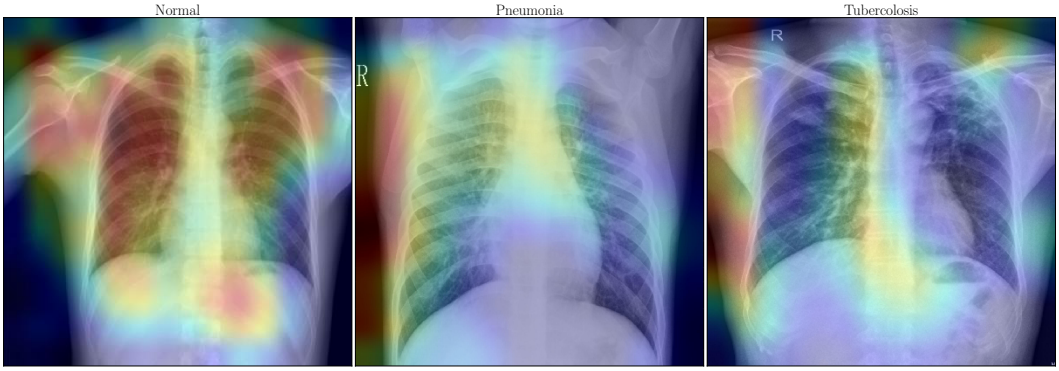


Figure above shows the results of the Grad-CAM algorithm applied to the ResNet. In the X-Ray images the importance of each pixel in the classification is enlightened with a color that goes from blue (not important) to red (very important). These three pictures are correctly classified by the network, however we can observe how just in the normal case the most considered pixels are actually inside the lungs, while in the other two cases, the network gives credit to pixels that are on the left side of the images, even outside the depicted human body. Having nearly zero medical background, little we can say about how to discriminate the different pathologies, but one might expect at least to look for discriminative pixels inside the lungs and not on the shoulders or on the side of the body as this network seems to do. It is likely that during the training the model discovered some patterns outside the lung area common to most Pneumonia images of the provided dataset, and the same happened for Tubercolosis ones. Despite the accuracy on our test set, this could prevent the network from recognizing new samples of ill lungs that do not present these patterns, while having clear signs of illness. These results make the network very difficult to explain, hence it is not suited in cases in which it is needed to build trust in clinicians and patients. If we add this fact to the poor performances with respect to the other models, this becomes easily the worst network we built.

12.2 SVM

Figure 14: SVM LIME results: Normal class

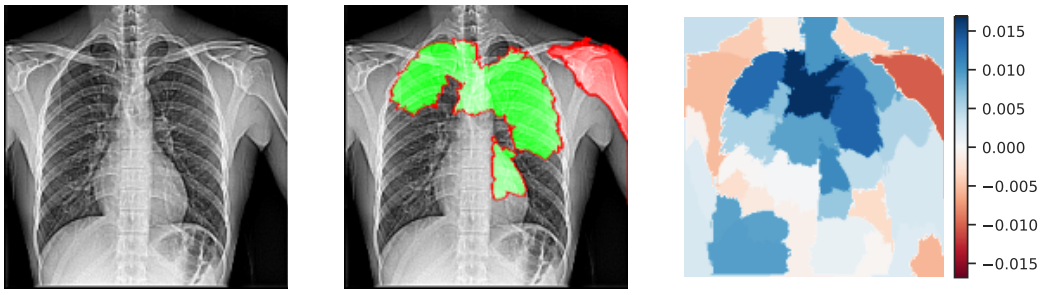


Figure 15: SVM LIME results: Pneumonia class

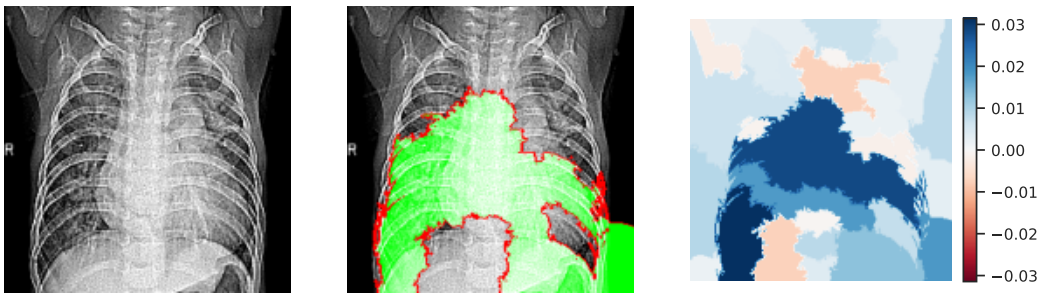
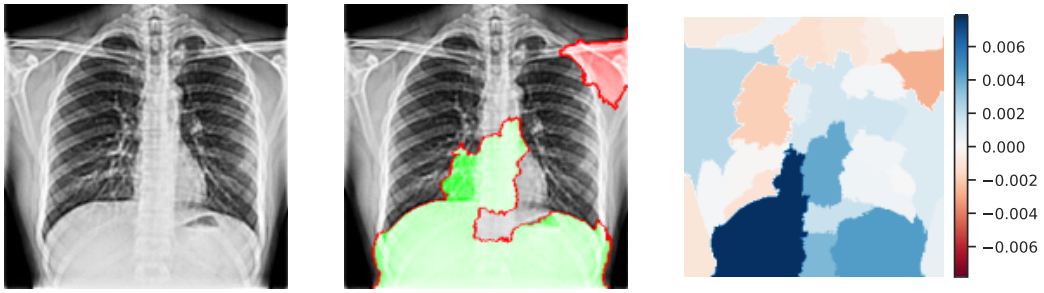


Figure 16: SVM LIME results: Tuberculosis class



The figures above show the results of the LIME algorithm applied on three X-Ray images correctly classified by the SVM model, one per each class. For each X-Ray there are three images: the first one simply shows the X-ray; the second one highlights in green the area which pixels positively contribute to the prediction of the label and in red the area which pixels negatively contribute to the prediction of the label; the third one shows how much each pixel of the X-Ray image contributes to the prediction: if the pixel contributes in a positive way, then it is in a shade of blue, whereas in case the contribution is negative, then the pixel is represented in a shade of red. The stronger is the color, the more the pixel influences the prediction. As we can see from these three examples, in the normal case the higher area of the lungs most influences the prediction, while in the other two cases the lower area of the chest influences the most. However, the most remarkable conclusion is that the pixels representing parts of the lungs helps in predicting all the three classes. So, this result is quite satisfactory since it appears that the SVM model classifies the X-Ray images in input based on some patterns in the lungs, which leads us to think that this model could be properly used by clinicians. On the other hand, the fact that in the case of the normal class the green area is in a very different position of the lungs with respect to the other two cases puts us in doubt as to what these patterns actually are and if the model's predictions are trustful.

12.3 EfficientNetV2B3

Figure 17: EfficientNetV2B3 One-vs-All Grad-CAM results

Figure above shows the results of the Grad-CAM algorithm applied to the EfficientNetV2B3 OvA model. Since the Ensemble is composed by 3 models, each of which specialized in predicting a particular class against the others, we performed the XAI analysis on each model predicting the class of interest. This means that, for example, the Pneumonia XAI output comes from the model that performs OvA classification focusing on that infection. All the three images are correctly classified by the networks, however different consideration can be done for each of them. Starting from the normal case, we can see that most of the relevant pixels are inside the lungs, even though some importance is also given to the background pixels. For the Tuberculosis sample, the corresponding network focuses on both upper and lower parts of the body, giving most of the importance to pixels in between the lower-right part of the body and the background. Finally the network responsible of classifying Pneumonia cases seems to provide the most trustful prediction, since all the attention is given to an unique part of the image, corresponding to the upper-left part of the body. Moreover this zone of focus is more aligned to the ones observable by the corresponding class images of the previous models' explanations, so we can conclude that the visual features of that image region can be useful to correctly predict the true class.

13 Discussion

Of all our experiments, the two that seemed most promising were the ones that used EfficientNetV2B3 as backbone, in particular, the EfficientNet Ensemble and the model that performed feature extraction with EficientNetV2B3 and classification with a SVM, and its ensemble version as well. They achieved overall better performance metrics with respect to the other models. Since we observed a highly imbalance in our dataset, the accuracy metric is not the best to use to assess performance, instead we focused on the f1-score, that combines precision and recall. The formula of this performance metric is the following:

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{7}$$

The proposed models achieved respectively **95.76%** overall f1-score for EfficientNetV2B3 Ensemble and **96.62%** overall f1-score for the SVM Ensemble. Going into more detail, and analyzing performances on the least represented class, the models achieved the following f1-scores in predicting *Tuberculosis* samples: **92.09%** for EfficientNet Ensemble and **93.62%** for SVM Ensemble. These values indicate that the models are good enough even in predicting examples from the class that was least represented.

In addition, they are also quite explainable, as we can see in the pictures of XAI section (12). Given the different natures of the models, we had to address Explainability in a slight different way. For what concerns EfficientNetV2B3 we plugged a Grad-CAM in order to visualize on the images the importance of pixels for the classification, while for the SVM we adopted the LIME algorithm, that can be applied even to models different from Neural Networks. As previously argued, due to our poor medical background, we cannot say whether the observed areas in the images are correct for the classification, but the Explainability algorithm applied to each model seemed to show that at least they are looking for some pattern inside the lungs, and this is consistent with the task.

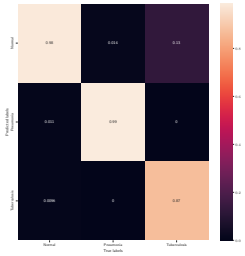
14 Conclusion

In conclusion, we tried a variety of different approaches in order to handle the task we were given. We initially observed that the provided dataset was characterized by high heterogeneity, since it was composed of images that were noisy, blurred or spoiled, hence we opted to clean it before the learning process.

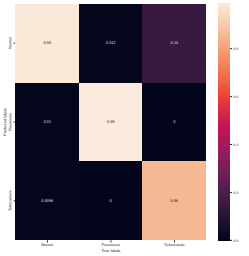
In our experiments, we noticed that the best way to perform feature extraction from images, in particular X-Rays, was to use Convolutional Neural Networks, since other feature extraction algorithms, such as *PCA*, did not achieve comparable performances. In particular it is better to import networks already trained on images of much bigger datasets such as *ImageNet* to perform some sort of feature extraction, and then fine tune the last layers to adapt them for our task.

Regarding Explainability, LIME and Grad-CAM came in our aid to try to show which parts of the images our models were taking into account while making predictions, in a second moment we could consult clinicians to analyze the outputs of Explainability algorithms and understand whether the models are sound enough to be used as tools to aid the doctors in recognizing the pathologies.

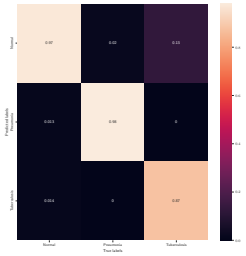
A Heathmaps



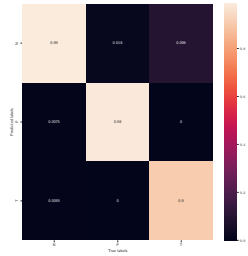
(a) ConvNextTiny & FFN w/o prepro.



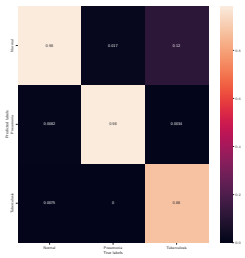
(b) ConvNextTiny & FFN w/ focal loss & class weights



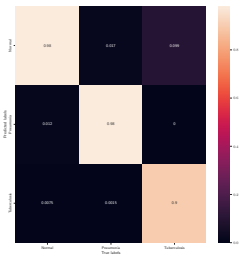
(c) ConvNextTiny & FFN w/ data aug



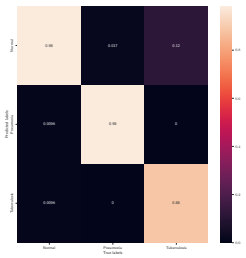
(d) EfficientNetV2B3 & FFN Ensemble



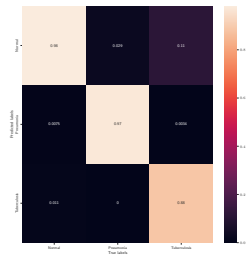
(e) EfficientNetV2B3 & SVM Fold 1



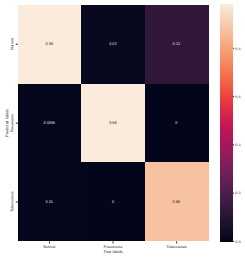
(f) EfficientNetV2B3 & SVM Fold 2



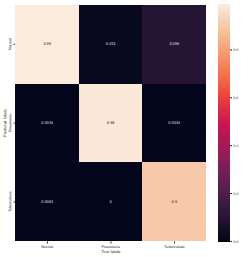
(g) EfficientNetV2B3 & SVM Fold 3



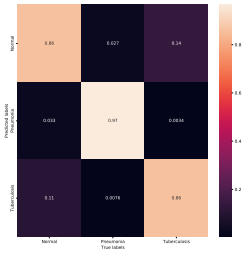
(h) EfficientNetV2B3 & SVM Fold 4



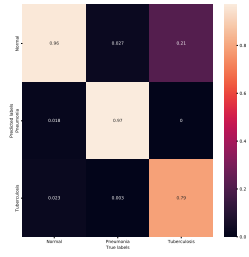
(i) EfficientNetV2B3 & SVM Fold 5



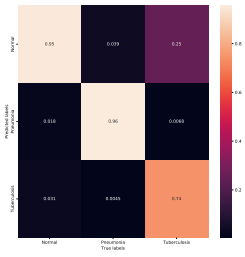
(j) EfficientNetV2B3 & SVM



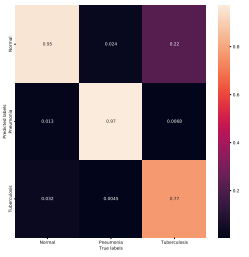
(k) ResNet cross-entropy



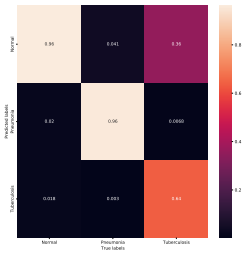
(l) ResNet focal loss



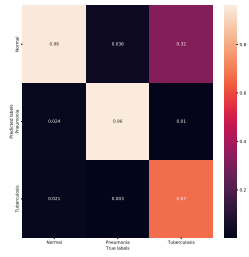
(m) ResNet Fold 1



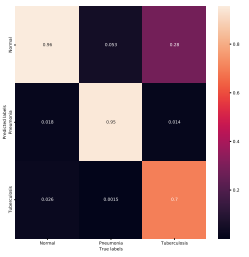
(n) ResNet Fold 2



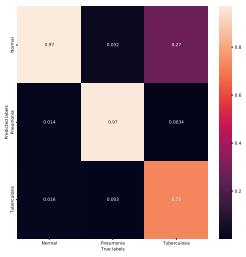
(o) ResNet Fold 3



(p) ResNet Fold 4



(q) ResNet Fold 5



(r) ResNet Ensemble

References

Aftab, Arya (2022). *Implementation of SVM in TensorFlow*. URL: https://github.com/AryaAftab/svm-tensorflow.

fchollet (2020). *Grad-CAM class activation visualization*. URL: https://keras.io/examples/vision/grad_cam/.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *CoRR*. URL: http://arxiv.org/abs/1512.03385.

Lin, Tsung-Yi et al. (2017). "Focal Loss for Dense Object Detection". In: *CoRR*. URL: http://arxiv.org/abs/1708.02002.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *CoRR*. URL: http://arxiv.org/abs/1602.04938.

Selvaraju, Ramprasaath R. et al. (2019). "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". In: *CoRR*. URL: http://arxiv.org/abs/1610.02391.

Wu, Ting-Fan, Chih-Jen Lin, and Ruby C. Weng (2004). "Probability Estimates for Multi-class Classification by Pairwise Coupling". In: *Journal of Machine Learning Research*. URL: https://www.csie.ntu.edu.tw/~cjlin/papers/svmprob/svmprob.pdf.

Zhou, Bolei et al. (2018). "Places: A 10 Million Image Database for Scene Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. URL: https://ieeexplore.ieee.org/document/7968387.