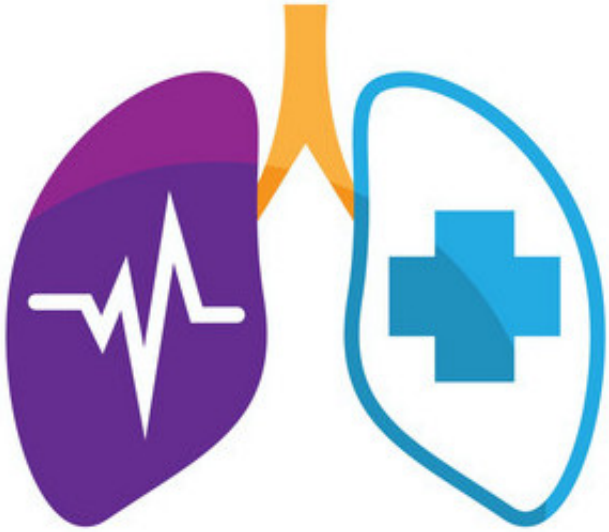
Applied AI Biomedicine

X-Ray images classifier

Federico Caspani, Vlad Cimpeanu, Sofia Martellozzo

January 2023



Contents

1 Introduction

Lung diseases are often associated with excruciating pain and suffering as they affect the breathing pattern of the patient due to suffocation and related symptoms. They are also one of the top causes of death worldwide and include pneumonia and tuberculosis diseases.

Pneumonia is an inflammatory condition of the lung that primarily affects the small air sacs called alveoli. It is usually caused by bacterial or viral infection, leading to air sacs in the lungs. Pneumonia symptoms typically include cough with phlegm, chest pain, difficulty breathing, fever and the severity of symptoms can vary. Pneumonia usually manifests as an area or areas of opacity on chest radiographs (CXRs).

Tuberculosis (TB) is a detrimental variant of lung infections and is caused by the *Mycobacterium tuberculosis* bacteria. Generally, lungs are the main target area of this disease, but it can also affect other parts of the body. TB is a contagious disease: when people infected with TB cough or sneeze, they transmit the disease-causing bacteria into the air. Only a small quantity of these germs is enough to effectively infect a healthy person. It is imperative to detect cases of TB as early as possible because if left untreated, there is a 70% chance of a patient dying within 10 years.

Therefore, early assessment and diagnosis can significantly reduce the life-threatening nature of lung diseases and improve the quality of life of suffering patients.

In modern medical image modalities, imaging tests are extremely powerful tools which can help doctors diagnose a range of conditions. One of the most commonly used image modalities is the chest X-ray radiographic image (CXR): a diagnostic tool that allows doctors to see the internal structures of the body without surgery and an earlier diagnosis of the disease.
Lung disease detection is currently performed through an examination of CXR images by a professional radiologist. However, there is a common human error that may be caused by the misreading of a CXR image, due to the complex anatomical structure of the chest. The rise of automation in the healthcare sector has allowed the introduction of supplementary tools to help radiologists to reduce the occurrence of this kind of errors. Deep-learning methods based on convolutional neural networks (CNNs) have exhibited increasing potential and efficiency in image recognition tasks, such as robotics, self-driving cars, and medical applications. Deep-learning models applied to CXR can achieve detection performance close to that of radiologists. Therefore, CNN models are used to help radiologists facing clinical decisions with more precise diagnoses and to minimize misreading.

In this project, we investigate machine learning-based methods for the classification of chest X-ray images.

2 Material and methods

In this section, we provide an overview of the dataset and the machine learning pipeline used to build our final model, with a detailed explanation of all the different models developed.

2.1 Dataset

The dataset used for model development and evaluation is composed of 15.470 CXR images of 12.086 different patients.
Compared to the natural RGB image, the CXR image is a grayscale image with one-channel information. The information of the CXR image is scanned by an X-ray, also called a radiograph. The X-ray sends radiation through the body. Areas with high levels of calcium (i.e., bones) block the radiation, causing them to appear white in the image. Soft tissues (i.e., lungs, heart, liver, muscle, and more) allow the radiation to pass through and appear in a range from grey to black on the image.
The size of the images provided is (400, 400, 1), where the values represent the image width, height and number of

channels.
From Figure 1 it is possible to notice that the data set is imbalanced: it contains the majority of samples without any diseases (N), a lower number of samples for pneumonia cases(P), and even fewer cases of tuberculosis (T).

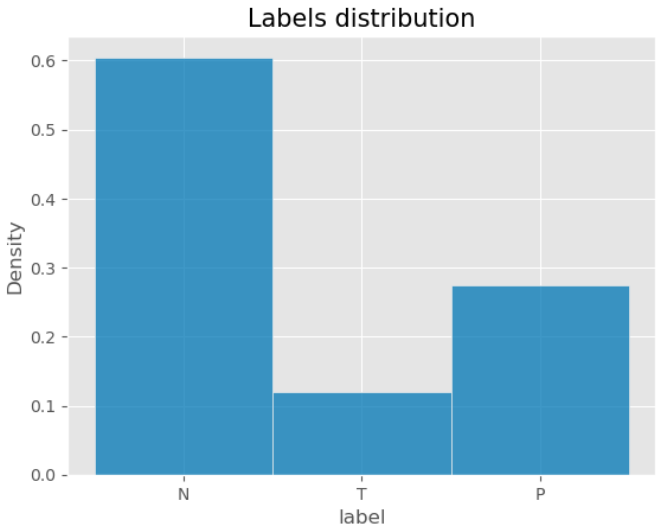


Figure 1: Images distribution.

Looking in detail at the images, we notice that a lot of them are noisy (Figure 2) as if an error occurred during the acquisition of the X-ray image. We decided to keep all of these images for two reasons: first, due to our lack of knowledge in the domain, we do not know if they contain relevant information to detect the disease, even if in a much more difficult way. Second, because we want our model to be trained on them as well, in order to be able to correctly classify bad quality images contained in the test set.

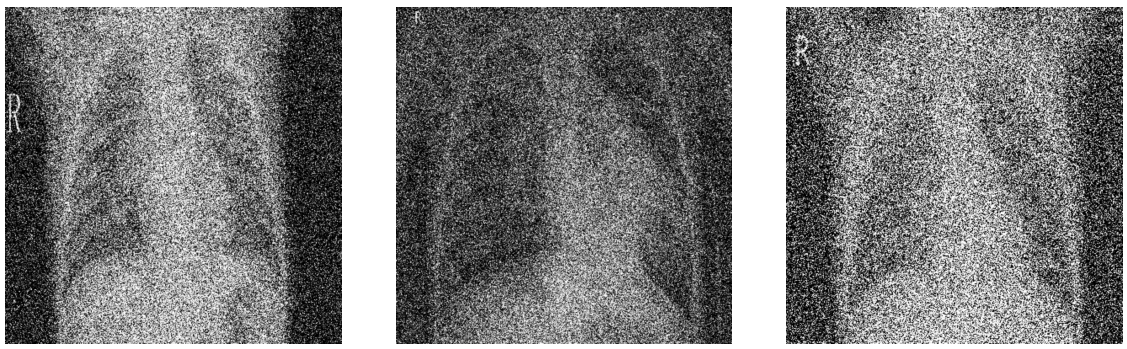


Figure 2: Example of noisy images.

2.2 Pipeline

First of all, we split our dataset into three sets: training ($\approx 70\%$ of the data), validation ($\approx 15\%$), and test set ($\approx 15\%$). Since we are dealing with an unbalanced dataset, we use a stratified split to ensure the labels' distribution is preserved in all the sets. Furthermore, as there are some patients with multiple images, we take care to assign all the images related to the same patient to the same set, to avoid any possible bias.

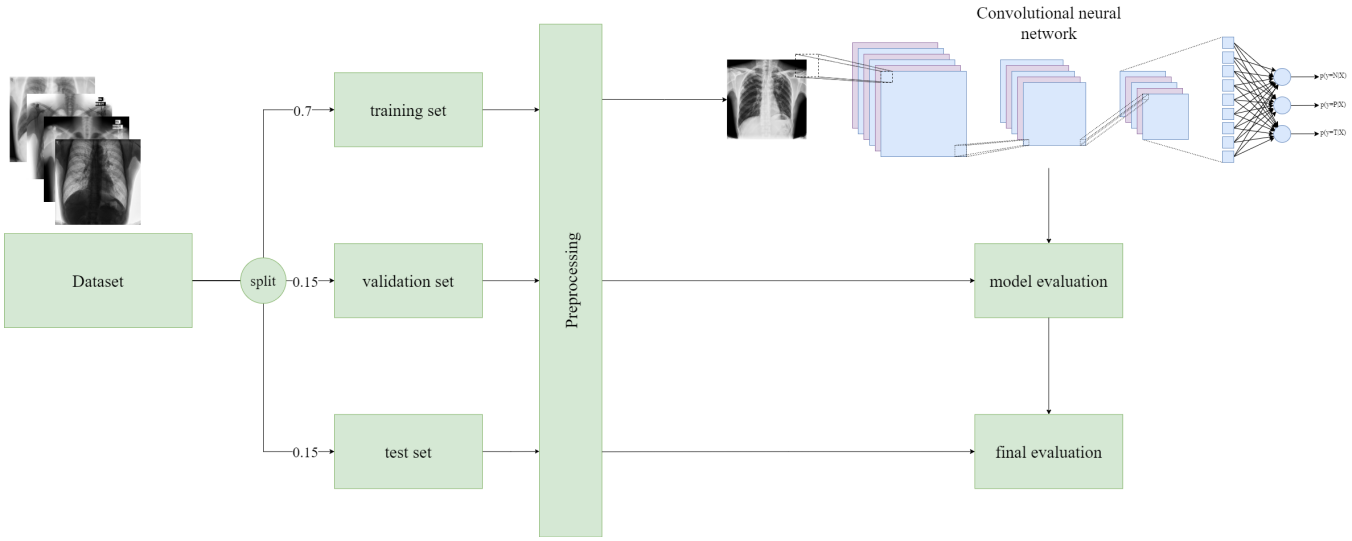


Figure 3: High-level view of the machine learning pipeline.

2.2.1 Preprocessing

In this section, we explain the preprocessing technique we apply, inspired by this paper [7].

Histogram equalization is a basic image processing technique that adjusts the global contrast of an image by updating the pixel intensity distribution of the image histogram. This technique enables areas of low contrast to obtain higher contrast in the output image. Essentially, histogram equalization works by:

1. Computing a histogram of image pixel intensities
2. Evenly spreading out and distributing the most frequent pixel values (i.e., the ones with the largest counts in the histogram)
3. Giving a linear trend to the cumulative distribution function (CDF)

The result of applying histogram equalization is an image with higher global contrast.
The possible drawback with histogram equalization is that it considers the global contrast of the image. This could lead to a loss of information if there are large intensity variations and histogram covers a large region (for example if both bright and dark pixels are present). This downside can be solved applying an algorithm called Contrast Limited Adaptive Histogram Equalization (CLAHE), the most widely used histogram equalization application that can be found in the medical field, resulting in higher quality output images. The algorithm simply works by dividing the image in small blocks called "tiles", and then applying histogram equalization to each one of these blocks. With the use of CLAHE, we enhance the contrast of X-ray images.

Gaussian Blur is a widely used effect in graphics software, typically to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent

screen, distinctly different from the bokeh effect produced by an out-of-focus lens or the shadow of an object under usual illumination. Gaussian smoothing is also used as a pre-processing stage in computer vision algorithms in order to enhance image structures at different scales.
Mathematically, applying a Gaussian blur to an image is the same as convolving the image with a Gaussian function which acts as a low-pass filter. The Gaussian blur is a type of image-blurring filter that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image.
The formula of a Gaussian function in two dimensions is the product of two Gaussian functions, one in each dimension:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution.
Values from this distribution are used to build a convolution matrix which is applied to the original image. Each pixel's new value is set to a weighted average of that pixel's neighborhood. The original pixel's value receives the heaviest weight (since it is placed in the origin of the Gaussian function) and neighboring pixels receive smaller weights as their distance to the origin increases. This results in a blur that preserves boundaries and edges better than other blurring filters.

Another important operation performed on the data before training our models is resizing the images. As said in the previous section 2.1, the dataset is composed of images (400, 400, 1). 400 is a really large dimension, not only to apply transfer learning (where resizing is mandatory in order to fit the input with the downloaded structure) but also with the model developed by hand (where it could generate difficulty and many delays in the training phase). Based on the model, we resize the images reducing their size to (256,256,1) or (299, 299,1). In some cases, we modify also the third dimension because pre-trained models require colored (RGB) images since they were trained on the 'ImageNet' dataset, not grayscale ones. How it is performed is explained in detail in the models' specifications 2.3.

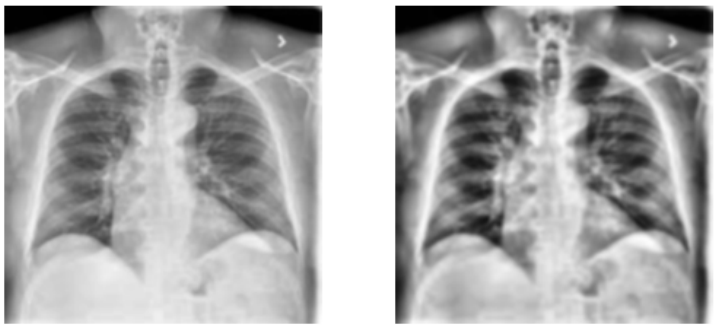


Figure 4: Example of preprocessing.

2.2.2 Data Augmentation

To mitigate the effect of the scarcity of tuberculosis images and to prevent overfitting, we apply different data augmentation techniques at training time, and some of them are common to all of our models proposed.
First, we perform a MixUp augmentation[2], which consists of applying a convex linear combination between two images belonging to the same batch:

$$MixUp(X_1, X_2, \lambda) = X_1(1-\lambda) + X_2\lambda,$$

where $\lambda \in [0,1]$.
As the two images might belong to different classes, we have to apply the same convex linear combination to their labels too, getting a soft label for the new sample.
By looking at the given images, we notice several images belonging to the tuberculosis class present some patches over them as we can see in Figure 5, therefore, we believe these patches may bias the model, indeed, it may find that a tuberculosis image is characterized by the presence of patches. For this reason, we decide to use a `DropBlock2D` layer at the beginning of each model to add random patches to every image of the training set.

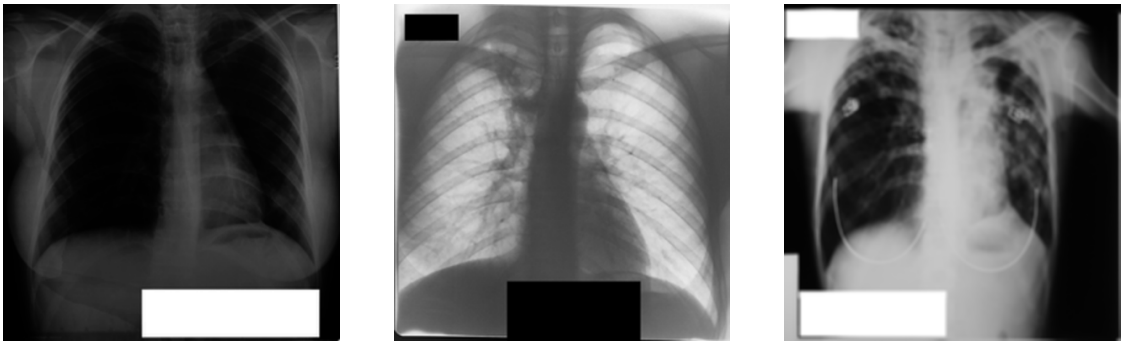


Figure 5: Example of tuberculosis images with patches.

Once we have prepared our data, we can train our models using the training set. Since this is a multiclass classification task, we try to minimize the categorical cross-entropy loss function. Actually, as our main goal is to maximize the f1-score for tuberculosis, we also tried to minimize the soft f1-loss function proposed here, but unfortunately, we were not able to reach interesting results.
Regarding this, we tune the categorical cross-entropy by weighting the loss based on the true label, more precisely, if the ground truth for a given sample is a non-frequent class (like pneumonia), the loss will weigh more with respect to a frequent class (like no symptoms class). Weights w_i are computed as follows:

$$w_i = \frac{1}{3c_i} \sum_{j=1}^{3} c_j,$$

where c_i are the occurrences of label i.
We decide to use weighted loss functions to deal with the unbalanced dataset due to hardware limitations: oversampling minority classes would lead us to a huge number of samples and so, untractable time computation, whereas we do not consider undersampling the minority class because neural networks are eager of data.
Finally, we select the model with the highest f1-score for tuberculosis on the validation set, and we perform the last assessment on the test set.

2.3 Models

In our work, we mainly focus on convolutional neural network methodologies, since these models have proven to perform much better than traditional machine learning approaches on image classification tasks.

2.3.1 Naive CNN

As first approach we decide to implement a naive convolutional neural network from scratch, using the following architecture: 4 convolutional layers with `ReLU` activation function, with 16, 32, 64 and, 128 filters, each one followed by a `BatchNormalization` layer and a `MaxPooling2D`. The last `MaxPooling2D` is followed by a convolutional layer with 3 filters, finally, we add a `GlobalAveragePooling2D` layer connected to a `Dense` layer with a `softmax` activation function and 3 neurons. This last layer will act as a classifier using the features extracted by the convolutions and will provide a probability distribution over the possible labels. Before sending an image to the CNN we scale it by dividing each pixel by 255.
Using a learning rate of 0.001 and 256 batch size, this first architecture leads us to a validation accuracy of 91.8% and the following metrics:

Metrics	No symptoms (N)	Pneumonia (P)	Tuberculosis (T)
Precision	0.931	0.972	0.845
Recall	0.960	0.958	0.746
F1-score	0.945	0.965	0.792

As we can see from Figure 6, the model struggles to distinguish tuberculosis from non-symptoms x-ray images, whereas pneumonia, is almost perfectly classified.

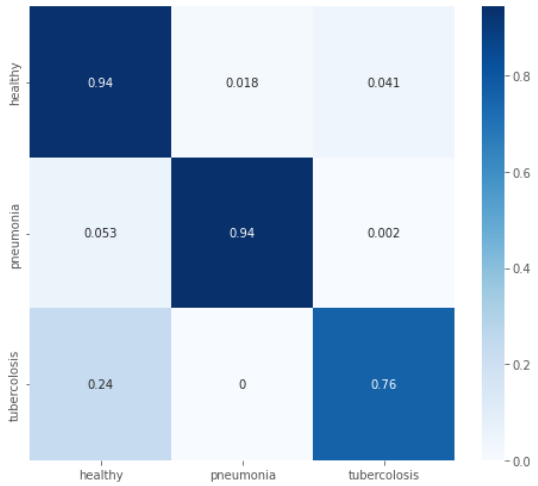


Figure 6: Confusion matrix for the naive CNN.

To improve the performances we try to change the thresholds used by our model to make a decision. By default, the model will select the label as:

$$\hat{y} = \arg\max_i (f(x))$$

thus, will choose the label with the highest value returned by the softmax function.

To maximize tuberculosis's f1-score, we can plot the precision vs recall curve for the binary classifier tuberculosis vs non-symptoms, assuming tuberculosis as the positive class, and find the threshold for which the curve is closest to the point (1, 1).

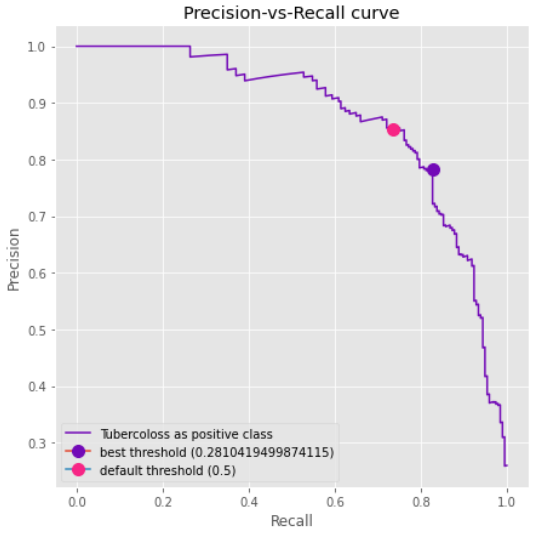


Figure 7: Precision vs Recall curve for the naive CNN.

From Figure 7, if pneumonia is not the label chosen by default, in order to maximize the f1-score, we should pick tuberculosis, if its conditional probability is at least 0.28.

2.3.2 DarkNet

In this section, we try to implement a revisited version of DarkNet [6]. In the cited paper, this architecture is used to detect Covid-19 from chest x-ray images, achieving decent results, thus, given the similarity between our task and the one faced by the paper, we belive this architecture to be promising.
As we can see from Figure 8, the main component is the DarkNet block which is composed of a `Conv2D` layer, followed by a `BatchNormalization` layer and a `LeakyReLU` activation layer.
The second main component is the `3xConv(in, out)` which is a sequence of 3 DarkNet blocks, respectively with a number of filters equal to `out`, `in`, `out`.
The last convolutional layer has no activation function and the number of filters is set to the number of classes, so, in our case 3, finally, the convolution is linked to a `Flatten` layer.

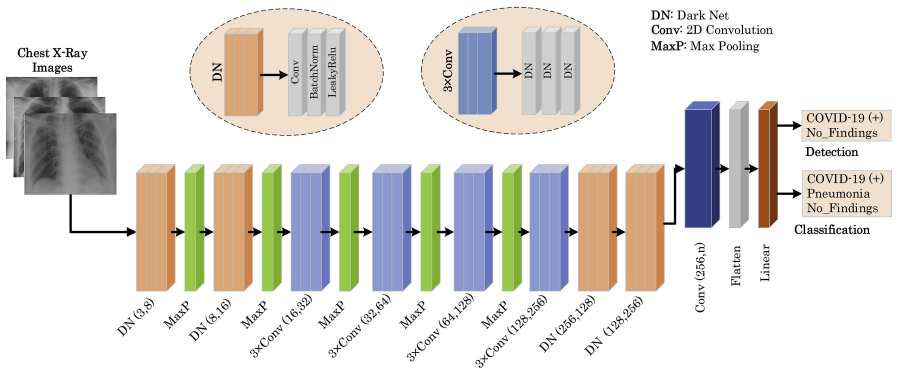


Figure 8: DarkNet architecture to detect Covid-19 [6].

By running our experiments with this Darknet version[6], we struggle to reach good performances both in the training and validation set. This leads us to think the convolutions are not powerful enough to extract meaningful features from our data, as matter of fact, this version differs from the original one[1] by the number of layers and number of filters per convolution (which are reduced).
Furthermore, the original version uses global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions, thus, this architecture should be able to capture more relevant features without letting the number of parameters explode.
Since none of the cited papers mention the preprocessing part, we decide to scale the images by dividing each pixel by 255.
With a learning rate of 0.003 and batch size of 64, our best experiment gets us with 93.8% accuracy on the validation set, and even in this case, from Figure 9, the model struggles to distinguish tuberculosis from non-symptoms, for this reason, we tune again the decision thresholds, reaching the following results:

Metrics	No symptoms (N)	Pneumonia (P)	Tuberculosis (T)
Precision	0.932	0.974	0.872
Recall	0.965	0.974	0.726
F1-score	0.948	0.974	0.792

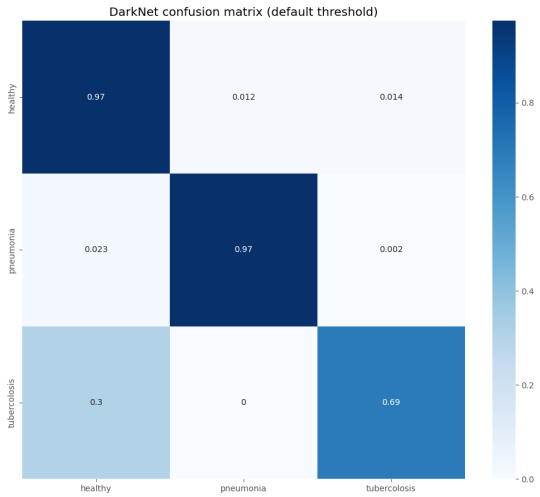


Figure 9: Confusion matrix for DarkNet-19 [1].

2.3.3 VGG19

To solve our problem we also try another approach: **Transfer Learning** and **Fine Tuning**.
Transfer learning is the process to take a network model that has already been trained for a given task and make it perform a second similar task. At first, the layers of a pre-trained model are downloaded by freezing them, to avoid destroying any of the information they contain during future training rounds. Some additional layers are added on top of the frozen ones with an output layer set to recognize the new classes (in our case 3: no symptoms, pneumonia, and tuberculosis). Then, these new layers are trained with the new dataset to take the most peculiar features from the front of the network and map them to the desired output classes.

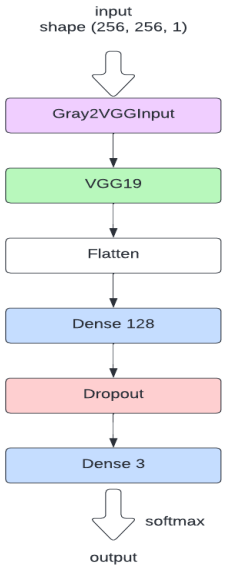


Figure 10: VGG19 model

The first pre-trained model we used is the **VGG19**, part of the `Keras Application` library, because of its size: 19 refers to the number of its layers with weights, counting a total of 20,024,384 parameters (much less compared to other models, such as inceptionV3[4]). Since the dataset is composed of a large amount of images, a lighter model results in easier training.
Concerning the top layers we attach a `Dense` layer of 128 neurons, after a `Flattening` layer, followed by a `Dropout` and the softmax for the output.
Since the VGG19 requires as input a coloured image (with 3 channel), we preprocess the input through an additional layer called `Gray2VGGInput`: it concatenate the original image 3 times and normalise it.
With the first training, we achieve an accuracy of 92% on the validation set with an initial learning rate of 0.001 and a batch size of 128.

Fine-tuning consists of unfreezing the entire model downloaded (or part of it), and re-training it on your data with a lower learning rate.
At first, we unfreeze the last 10 layers of the VGG19 and we reduce the learning rate to 10^{-4}, achieving really good results: 96% of accuracy on the validation set.
Finally, we unfreeze all the remaining layers and train the model with 10^{-5} as learning rate: in this case, the improvement was minimal, achieving almost the same accuracy as the previous step.

Metrics	No symptoms (N)	Pneumonia (P)	Tuberculosis (T)
Precision	0.962	0.962	0.948
Recall	0.974	0.991	0.832
F1-score	0.968	0.976	0.886

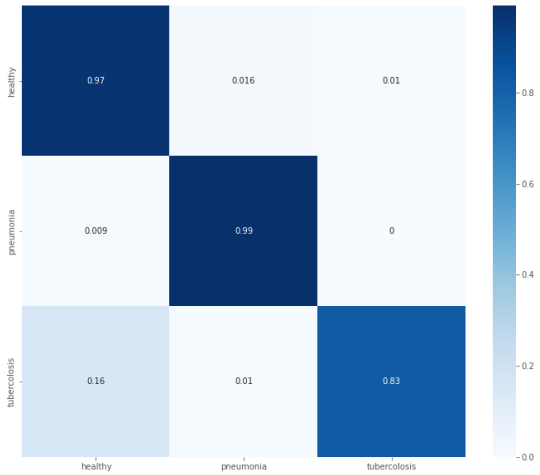


Figure 11: Confusion matrix on validation set for VGG19.

2.3.4 CoroNet

Taking inspiration from another paper [5] we tried to replicate and adapt the method presented in the study. Also in this case the solution follows the transfer learning and fine tuning approach and in this case is based on the **Xception** model. Xception, which stands for Extreme version of Inception (its predecessor model), is a 71 layers deep CNN architecture pre-trained on ImageNet dataset. Xception uses depthwise separable convolution layers with residual connections instead of classical convolutions.

Depthwise Separable Convolution replaces classic n x n x k convolution operation with 1 × 1 x k point-wise convolution operation followed by channel-wise n x n spatial convolution operation. This way the number of operations are reduced by a factor proportional to 1/k. Residual connections are 'skip connections' which allow gradients to flow through a network directly, without passing through non-linear activation functions and thus avoiding the problem of vanishing gradients. In residual connections, output of a weight layer series is added to the original input and then passed through non-linear activation function as shown in.

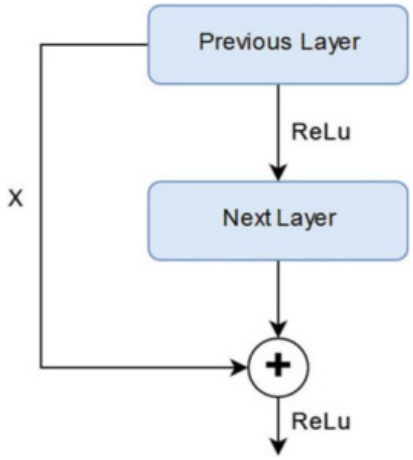


Figure 12: CoroNet sckipped connection

On top of the Xception network we added a `Dense` layer of 256 neurons, a `GAP` layer, a `Dropout` layer and then a `Softmax` layer for the output. This configuration was chosen after some attempts as it proved to be the best ensuring excellent performance and preventing network overfitting.
In order to fully exploit the power of Xception we performed some specific preprocessing on the input images. For starting, the cited model is optimized for images of size 299X299 so we reshaped the original images.
Also, as in the case of VGG19, Xception works with RGB images so we modified the input images through the OpenCv function "cvtColor" and the "COLOR_GRAY2RGB" parameter (again from OpenCv). In the end we applied the specific Xception preprocess which scales the input pixels between -1 and 1 before passing the images to the network.

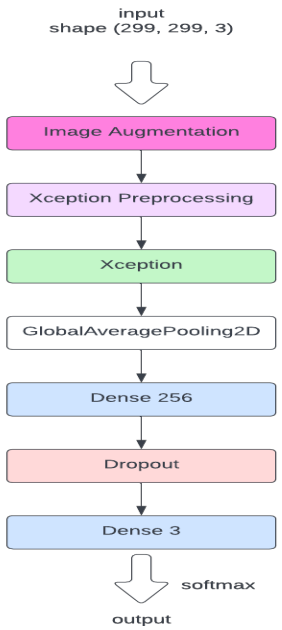


Figure 13: CoroNet model

Regarding the training of the model we first performed transfer learning, training the model with all the layers of the Xception model frozen for 120 epochs, using a learning rate of 0.0001 and a batch size of 120 images. We reached an accuracy of 82% on the train set and of 88% on the validation set. Then we fine tuned the model unfreezing all the layers for others 30 epochs and keeping the same hyperparameters, until we reached an accuracy of 94% in training and of 97% in validation.

The final results are reported in the table below and in Figure14

Metrics	No symptoms (N)	Pneumonia (P)	Tuberculosis (T)
Precision	0.956	0.988	0.97
Recall	0.989	0.984	0.822
F1-score	0.972	0.986	0.890

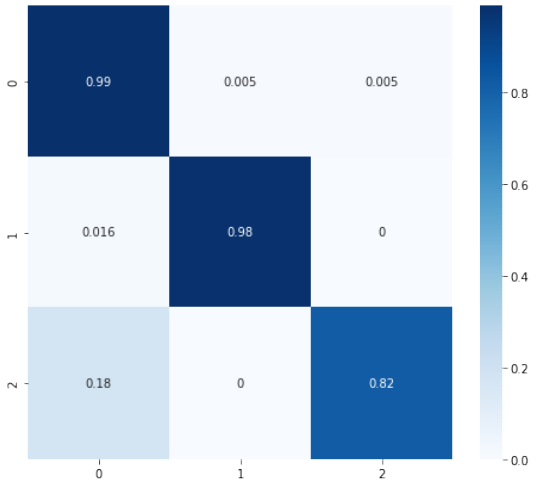


Figure 14: Confusion matrix on validation set for CoroNet

3 Results

To wrap up, below there are tables summarizing the results gotten by each model on the validation set and all the confusion matrices::

model	validation accuracy	f1 score (N)	f1 score (P)	f1 score (T)
CNN	0.918	0.943	0.965	0.8
DarkNet	0.938	0.948	0.974	0.792
VGG19	0.960	0.964	0.975	0.883
CoroNet	0.966	0.972	0.986	0.890

model	precision(N)	precision(P)	precision (T)	recall (N)	recall(P)	recall(T)
CNN	0.931	0.972	0.845	0.960	0.958	0.746
DarkNet	0.932	0.974	0.872	0.965	0.974	0.726
VGG19	0.962	0.962	0.948	0.974	0.991	0.832
CoroNet	0.956	0.988	0.970	0.989	0.984	0.822

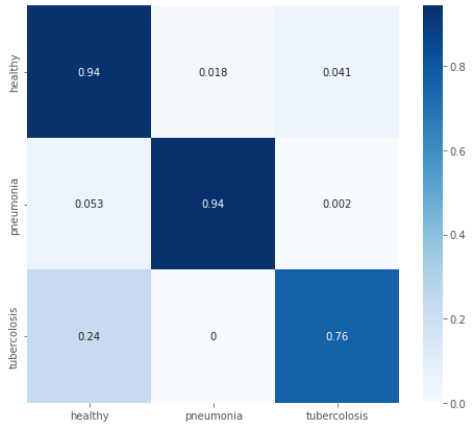


Figure 15: Confusion matrix of CNN

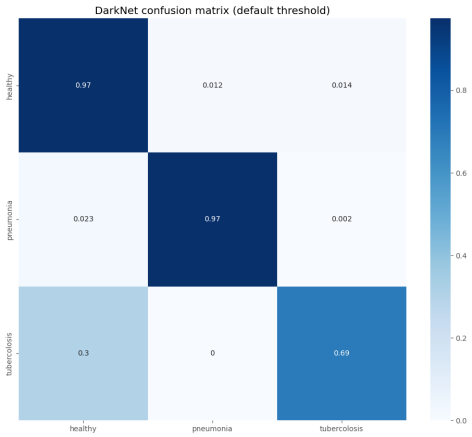


Figure 16: Confusion matrix of Darknet

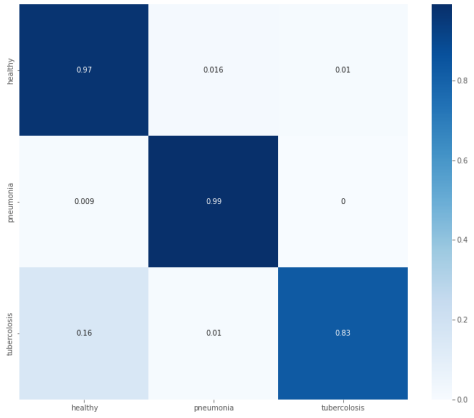


Figure 17: Confusion matrix of VGG19

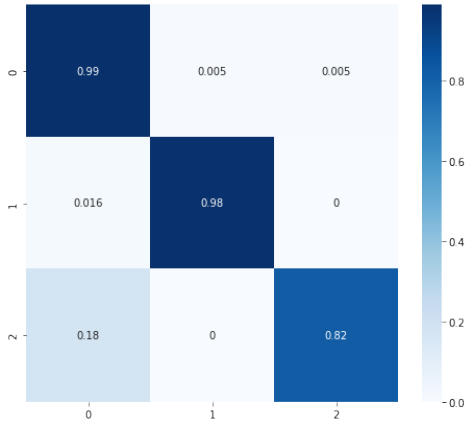


Figure 18: Confusion matrix of coroNet

The first consideration we can make from the confusion matrices is all models, more or less, perform well in detecting pneumonia, furthermore, pneumonia and tuberculosis are almost never confused, which is good, since wrong drugs may aggravate the patient's conditions. A more interesting but not comforting insight is all models have some problems recognizing tuberculosis; indeed, there are a few tuberculosis samples classified as no-findings (false negatives) which is not ideal in medical applications.
As we can see, CoroNet outperforms the other models, although VGG19 presents similar performances, for this reason, we choose CoroNet as our best model and make a final assessment using the test set.
On the test set, CoroNet achieves a 97% accuracy and a 0.91 f1-score for tuberculosis. Below we can find other interesting insights on the test set:

Metrics	No symptoms (N)	Pneumonia (P)	Tuberculosis (T)
Precision	0.972	0.978	0.943
Recall	0.980	0.985	0.887
F1-score	0.976	0.982	0.914

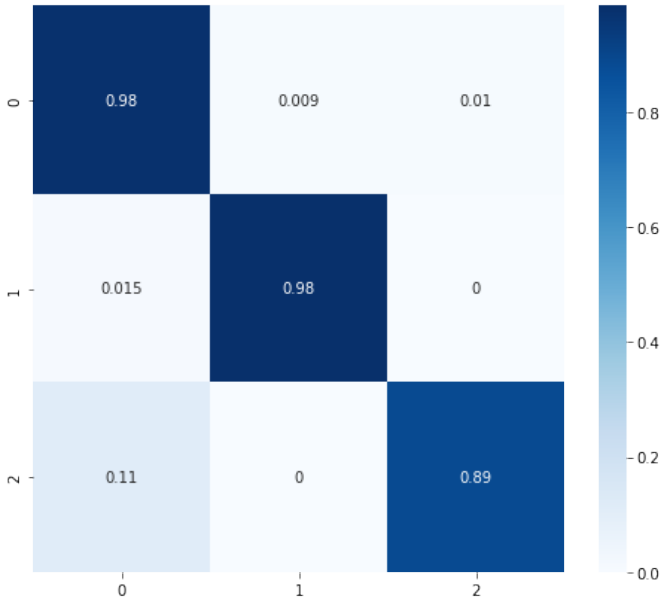


Figure 19: Confusion matrix for Coronet on the test set

4 Explainable AI

In this section, we try to interpret the results of our models and understand if these models can be trusted.
Since deep neural networks are black box models, it is impossible to explain the model just by looking at its parameters. For this reason, we try to implement post hoc methods to interpret the predictions performed by our models.

4.1 Occlusion method

First, we try a naive but easy-to-understand approach: the occlusion method. The input image is systematically and consistently modified by occluding, or blocking out, a rectangle of the same size and color. Each time the model analyses the image with a section occluded, its prediction changes.
The idea behind this is that once the patch is placed over the area responsible for the correct prediction, the model should no more be able to predict the correct label, for instance in the case of a tuberculosis CXR, once the patch overlaps the interesting area, the model should predict a different label from the correct one(in our case no-symptoms).

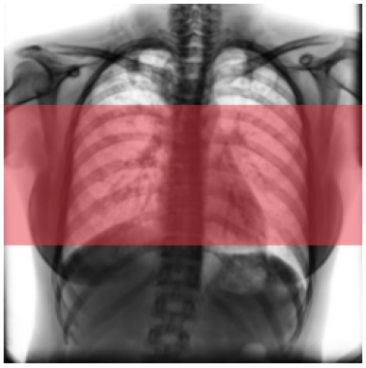


Figure 20: Occlusion method on tuberculosis image.

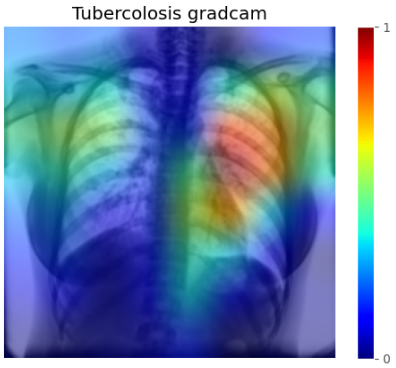


Figure 21: Gradcam method on tuberculosis image.

One main problem with the occlusion method is it is difficult to find the right sizes for the patch, as they depend on the size of the interesting area in the image. This is not the only problem, it might be the case that the disease can be spotted in multiple places of the same image, thus, occluding one area will not impact the final result.
One way to solve this issue may be to use horizontal rectangles having a width equal to the image's width, but, in this case, we will lose a lot of information about the position of the interesting area.
Figure 20 is a clear example of the issue just mentioned, as matter of fact, this image presents signs of tuberculosis on both lungs(according to the Gradcam[3] as Figure 21 suggests), but from the analysis provided by the occlusion method, we are not able to understand it.

4.2 Gradcam

Not satisfied by the occlusion method's results, we try a more sophisticated technique such as Gradcam[3].
This approach uses the gradients of any target concept (say 'tuberculosis' in a classification network) flowing into the final convolutional layer(which is supposed to contain the highest level features) to produce a coarse localization map highlighting the important regions in the image for predicting the concept.
The localization is not fine since it relies on the last convolution layer's activations, which have a lower resolution than the input image, due to the presence of max-pooling layers.

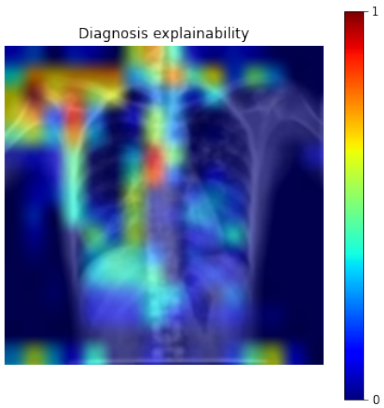


Figure 22: Gradcam method with cnn.

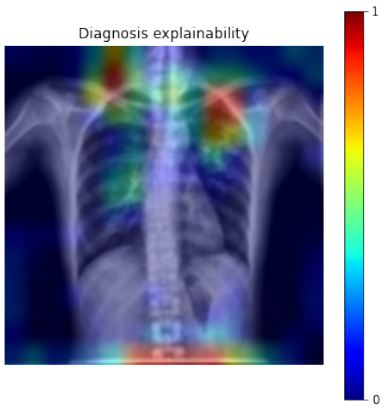


Figure 23: Gradcam method with VGG.

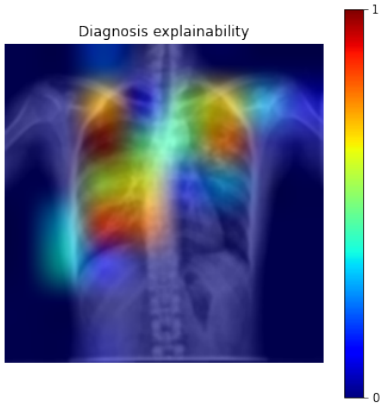


Figure 24: Gradcam method with DarkNet.

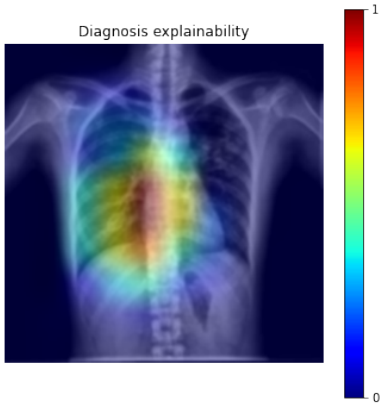


Figure 25: Gradcam method with coroNet.

As we can see from the figure above, the naive CNN and VGG19, despite their good performances, struggle to find the right features, as they seem to highlight random parts of the images.
Instead, DarkNet and CoroNet highlight reasonable parts of the body, nevertheless, DarkNet highlights more areas than CoroNet. Since we do not have any prior knowledge, it is unclear to us if DarkNet is also spotting useless features, or, CoroNet cannot extract all the meaningful features, for this reason, an evaluation carried out by experts should be performed.

5 Conclusions

In our work, we compared the performance of four different neural network architectures for detecting tuberculosis and pneumonia from chest x-ray images: a CNN from scratch, the DarkNet architecture, VGG19, and CoroNet. Our results showed that both Coronet and VGG19 achieved high accuracy in the tuberculosis detection task, but VGG19 could not extract meaningful features from the images.
On the other hand, the CNN from scratch and the DarkNet architecture performed relatively poorly in detecting tuberculosis, even though they showed high accuracy. In general, all models showed some limitations in detecting tuberculosis, probably because of a lack of images.
Overall, these findings suggest that both Coronet and VGG19 can be effective options for this application, but VGG19 may not be suitable for understanding the underlying characteristics of the diseases, hence, Corenet should be preferred. Due to hardware limitations, we avoided the use of k-fold cross validation, but we believe further works could explore the use of this method to get more robust results.

References

[1] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. DOI: 10.48550/ARXIV.1612.08242. URL: https://arxiv.org/abs/1612.08242.

[2] Hongyi Zhang et al. *mixup: Beyond Empirical Risk Minimization*. 2017. DOI: 10.48550/ARXIV.1710.09412. URL: https://arxiv.org/abs/1710.09412.

[3] Ramprasaath R. Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision* 128.2 (Oct. 2019), pp. 336–359. DOI: 10.1007/s11263-019-01228-7. URL: https://doi.org/10.1007%2Fs11263-019-01228-7.

[4] Sohaib Asif et al. "Classification of COVID-19 from Chest X-ray images using Deep Convolutional Neural Network". In: *2020 IEEE 6th International Conference on Computer and Communications (ICCC)*. 2020, pp. 426–433. DOI: 10.1109/ICCC51575.2020.9344870.

[5] Asif Iqbal Khan, Junaid Latief Shah, and Mohammad Mudasir Bhat. "CoroNet: A deep neural network for detection and diagnosis of COVID-19 from chest x-ray images". In: *Computer Methods and Programs in Biomedicine* 196 (2020), p. 105581. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2020.105581. URL: https://www.sciencedirect.com/science/article/pii/S0169260720314140.

[6] Tulin Ozturk et al. "Automated detection of COVID-19 cases using deep neural networks with X-ray images". In: *Computers in Biology and Medicine* 121 (2020), p. 103792. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2020.103792. URL: https://www.sciencedirect.com/science/article/pii/S0010482520301621.

[7] Martyna Tarczewska Agata Giełczyk Anna Marciniak and Zbigniew Lutowski. "Pre-processing methods in chest X-ray image classification". In: *PLoS One* 128.2 (Apr. 2022), pp. 336–359. DOI: 10.1371/journal.pone.0265949. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8982897/.