Applied AI in Biomedicine

X-Ray images classification

Authors:
La Greca Michele Carlo (10864460)
Nunziante Matteo (10670132)

Professor:
Prof. V. Corino

January 31, 2023

Contents

1 Introduction

Pneumonia and tuberculosis (TB) are both serious lung infections that can cause severe illness and even death. Pneumonia is an infection of the lung tissue that can be caused by a variety of microorganisms, including bacteria, viruses, and fungi. TB, on the other hand, is a bacterial infection that primarily affects the lungs, but can also affect other parts of the body.

Both pneumonia and TB can be difficult to diagnose, especially in areas where access to diagnostic tools such as X-ray imaging may be limited. Radiologists often rely on visual cues such as the presence of shadows, opacity, and other features in the lungs. However, these visual cues can be subjective and may not always be reliable indicators of the presence of infection. This is where deep learning techniques can play a critical role by automating the diagnosis process and providing more objective and accurate results.

Deep learning is a subfield of machine learning that has been widely used in image classification tasks, including medical imaging. The use of deep learning algorithms can improve the accuracy of the diagnosis of lung infections such as pneumonia and TB. By training a convolutional neural network (CNN) on a large dataset of X-ray images of the lungs, the network can learn to identify patterns in the images that are indicative of pneumonia or TB. The typical deep learning pipeline will be used, focusing on 3 different CNN architectures. Tensorflow and in particular Keras framework will be used. After the classification task, an Explainable AI analysis will be implemented. XAI analysis on image classification using deep learning and convolutional neural networks can be useful especially because it can help make the model's decision-making process more transparent and interpretable, which can be important for building trust and gaining acceptance of the model in certain applications since typically DL methods are black-boxes and they are hard to explain by looking only at the structure of the model.

2 Materials and Methods

2.1 Data

The input data consists of 15470 X-ray images of various sizes: 9354 images refer to normal cases, 4250 images to Pneumonia, and 1866 images to TB. The images are quite unbalanced, and this aspect will be taken into consideration using class weights during the training. The name of each image is made of two parts: the first part identifies the patient of the image, and the second part is the id of the image for that specific patient. Therefore, a patient can have multiple images referred to her, and these can also be of different classes. In the beginning, the images and the CSV file with the target for each image were in the same folder. In order to make the task easier, a folder for each class (N, P, T) has been created, and the images with that specific class have been moved to the corresponding folder.

Figure 1: X-ray of a normal case

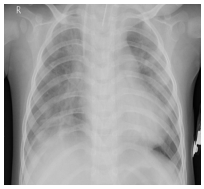


Figure 3: X-ray of a Pneumonia case

Figure 2: X-ray of a Tuberculosis case

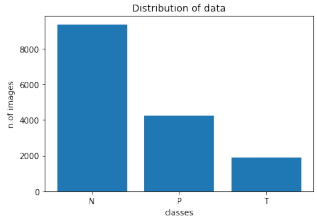


Figure 4: Distribution of the classes

2.2 Preprocessing

2.2.1 Splitting

In order to train a CNN, the dataset has been divided into 3 sets. The training set will be used to train the

model. The validation set and the test set will not be used for the training. The validation set will be used for tuning the hyperparameters, and for the model selection. The test set will be used at the end of the model development to perform a final test of the model. The first splitting is made by dividing the data in train, validation, and test set, with a ratio of 80%, 15%, and 15%. This split has been made by images and not by patients. In fact, it can happen that a patient has multiple images of the same class and that they go in different splits, causing overfitting since they are the same. Therefore the splitting has been done by patients, meaning that the images of the same individual belong to only one partition of the data. By exploring the images, this problem involved a few images, but still, it was theoretically wrong for the scope of the project and it could lead to an increase in overfitting. The training set contains 10992 images: class N has 6647 images, class P has 3025 images, and class T has 1320 images. The validation set contains 2487 images: class N has 1506 images, class P has 678 images, and class T has 303 images. The test set contains 1991 images: class N has 1201 images, class P has 547 images, and class T has 243 images. The splitting has been performed with the stratified approach, keeping the distribution of the 3 classes unchanged in the 3 splits.

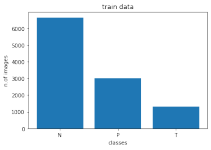


Figure 5: Distribution of the training data

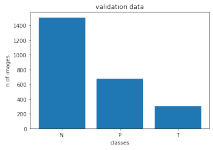


Figure 6: Distribution of the validation data

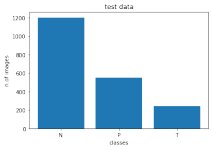


Figure 7: Distribution of the test data

2.2.2 Image Size

Images have different sizes. Visualizing the distribution of sizes, 400x400 is the most frequent size. Moreover, it is convenient to reshape all the images to this size during training to reduce the complexity of the model in terms of dimensionality.

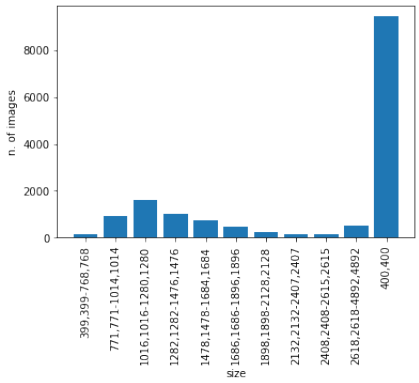


Figure 8: Distribution of image sizes

2.2.3 Data Generator

Data generators in Keras are useful because they allows to efficiently feed large amounts of data into a model during training. Without using a data generator, loading all of the data into memory at once would be needed, which can cause issues with a large dataset. Data generators allow the loading of the data in small batches, reducing memory usage and allowing training on larger datasets. How many images will be generated depends on the size of the batch and the input data set that contains a specific number of inputs. If each batch has few training images, overall the epoch will see more images than using big batches. This is due to the fact that, to take a batch, shuffle is used, and using a small batch we have a smaller probability to have in that batch some data that was already present in some previous batches. By using a small batch, the model will be trained with more distinct and less repeated images. First, Keras image_data_generator[7] has been used for the generation of the batches containing the images. Then, in order to apply prefetching, image_data_generator has been replaced with image_dataset_from_directory[8]. Prefetch allows prefetching a certain number of batches of data so that they are ready to be used by the model when it needs them. This can improve the performance of the model by reducing the time spent waiting for data to be loaded. Validation data and test data were not shuffled during the generation, in order to keep the data in the same order as it is a representative sample of the true distribution of the data.

2.2.4 Data Augmentation

Data augmentation[5] is a technique used to increase the diversity of the training set by applying random (but realistic) transformations. We used different augmentations:

- **Random flip**[11]: flip the input images vertically end horizontally;
- **Random zoom**[13]: randomly zoom in or out on each axis of an image independently;
- **Random translation**[12]: randomly translate the images on each axis;
- **Random constrast**[10]: randomly adjusts contrast during training;
- **Cropping**[4]: it crops along spatial dimensions. In our case, we reduced images to a size of 400x300.

We implemented these augmentation techniques using special layers inside the model before the feature extraction network and they have been applied only at training time, meaning that at inference time the input images have not been modified.

Exploring the dataset we noticed that some of the images are noisy and thus they can be considered as a sort of augmentation. Since these data were used during the training they helped the model to be also invariant to noise.

2.3 Models

We proposed 3 different models: scratch, InceptionResNetV2, and EfficientNetV2B2 models. The model named "scratch" is a model created by us while the last 2 are models fine-tuned from well-known public networks.

2.3.1 Scratch model

The first NN to be implemented was a network from scratch. A good practise which it good to start with is the fact that the number of channels should be low in the first layers so that they detect low-level and local features, and more filters in the next layers in order to capture many complex shapes. The final version of this network had 32,517,251 parameters. The model has 8 convolutional layers with an increasing filter depth: 128, 128, 256, 256, 512, 512, 1024, and 1024. Between every 4 layers, a dropout with a probability of 0.3 was inserted. After a GAP, a dense of 1024 neurons with L1 and L2 regularization is placed before another dropout layer with probability 0.3 and the output layer.

2.3.2 InceptionResNetV2 model

InceptionResNetV2[9] was used as a feature extractor by using transfer learning. The original network has inception blocks that are increasing the depth of the network by concatenating the input with the convolutions of different types of filters. It also implemented the residual connections. On top of the feature extractor, a FC part composed of a GAP, a dense layer of 1024 neurons, and a dropout with 0.3 as prob- ability has been placed. The feature extractor part was trained with imagenet dataset. The network consists of 55,853,155 parameters.

2.3.3 EfficientNetV2B2 model

EfficientNetV2B2[6] was used as a feature extractor by using transfer learning. Also, this network was trained on the Imagenet dataset, thus it has been necessary to retrain the whole network. At the bottom of the EfficientNetV2B2 network, a classic neural network classified has been placed. The latter is made of 5 dense layers with, in order, 512, 256, 192, 48, and 3 neurons. These layers have been divided using dropout layers with a probability of 0.2. In the first 4 layers, the Swish activation function [19] has been used, while in the last one, it has been used Softmax activation function [18]. Totally, this model is made of 9,680,865 parameters. The complexity of this model with respect to the first 2 models is very low and that's why the only regularization used was the dropout.

2.3.4 Ensemble

We tried to combine the previous models creating a so-called ensemble. We tried 2 different approaches: **the classic approach** and **the confidence approach**.

2.3.5 Classic ensemble

This method consists in simply averaging the predictions of each model and selecting the predicted task according

to the mean of the predictions. An alternative method is majority voting.

2.3.6 Confidence ensemble

First of all, we introduced the definition of **confidence** as how much a model is sure about a single prediction, which in our case can be summarized as the probability with which the model is predicting a certain class. This approach can be done only if more than one non-trivial predictor is available. The basic idea consists in feeding the first model with all the images but considering valid only the prediction done with a probability over a predefined confidence threshold. Then, the non-yet-classified images are fed to the next model, and as before only the high-confidence predictions are kept. The larger the number of available models the larger the number of high-confidence predictions.

2.4 Training

2.4.1 Shuffling Data

Shuffling the training data is beneficial because it helps to reduce overfitting during the training. When training a neural network, the model can easily memorize the training data if it is presented in a consistent order. This is because the model will learn to recognize patterns in the data that are specific to the ordering rather than general patterns in the data. By shuffling the training data, the model is exposed to a wider variety of patterns in the data, which can help to improve its ability to generalize to new, unseen data. Additionally, it can also prevent the model from getting stuck in a local minimum during the optimization process. It is generally not recommended to shuffle the data in the validation set, as it is used to evaluate the performance of the model on unseen data. Shuffling the validation set would make it difficult to accurately compare the model's performance to a fixed benchmark. It's better to keep the validation set in the same order as it is a representative sample of the true distribution of the data.

2.4.2 Callbacks

The training has been performed using EarlyStopping and ReduceLROnPlateau callbacks. A callback is an object that can perform actions at various stages of training [3]. EarlyStopping monitors the value of a metric after each epoch, in our case the metric is the validation accuracy, and it stops the training when the metric is not increasing anymore after K steps of patience. ReduceLROnPlateau, instead, reduces the learning rate when the metric to monitor has stopped improving.

2.4.3 Optimizer and Loss function

All the models have been trained using the Adam optimizer with Categorical Cross entropy as loss function.

2.4.4 Class weights

Class weights are used in training CNNs and other machine-learning models to address the class imbalance, which occurs when some classes in the dataset have significantly more samples than others. This can cause the model to be biased towards the majority classes, resulting in poor performance on the minority classes. By assigning class weights, the model is penalized more for misclassifying minority class samples, which helps to balance the class distribution and improve overall performance. Class imbalance should be corrected during the training phase, and should not be applied to validation and test data, to keep this data unchanged. In our case, we used a class weight that is proportional to the number of images of that class with respect to the total amount, resulting in the following weights: N: 0.551, P: 1.211, T: 2.776.

2.4.5 Training characteristics

The scratch network has been implemented sequentially by changing the structure and measuring the validation performances. When the model reached around 32 million of parameters, using 8 convolutional layers each followed by a max pooling, a problem was raised: after the 10th epoch, the performances dramatically dropped by 30%. Firstly, the complexity is high and since the very first epochs, the performances of the model were high. Adam is used as an optimizer, and the default learning rate is 0.001. But since the performance was very high from the very first epochs, the LR in those epochs was still high and unstable, being too early the adaptive method of Adam. This caused a drop. Initializing a

lower LR solved the problem. Using dropout, L1 and L2 regularization, and class weight, overfitting[1] decreased, bringing the accuracy on the validation around 0.94, and on the test around 0.90. On average, each version was trained for 25 epochs. This network has been trained without data augmentation and with just early stopping as callbacks. The learning rate used were 0.0001.

InceptionResNetV2 and EfficientNetV2B2 have been trained using transfer learning and fine tuning. InceptionResNetV2 has been gradualy fine tuned, unfreezing and training every time more and more parameters. Infact, with few trainable parameters the performances were low. This is possibly due to the fact that the feature extractor, even if it was trained using Imagenet dataset, suffered the specificity of the problem. Then fine tuning was performed. Given the complex structure of the feature extractor in the second and in the third model, augmentation of the data needed to be applied to reduce overfitting. In order to speed-up the learning of the parameters we used ReduceRLOnPlateau callback in order to start with an high learning rate (0.001) and then gradually reduce it by a factor of 0.5 each time the model seems to converge to a plateau.

Dropout regularization has been used in all the three models, while weight decay has been used only in the first two models since they have have a very large number of parameters, maybe too much for that specific situation.

3 Results

3.1 Metrics

The metrics[17] used in order to evaluate the model performances are confusion matrix, accuracy, precision, recall, and F1-score.

- **Confusion matrix**: tells the number of points which have been correctly classified and those which have been miss-classified for each class.
- **Accuracy**: $Acc = \frac{tp+tn}{N}$; fraction of the samples correctly classified in the dataset;
- **Precision**: $Pre = \frac{tp}{tp+fp}$; the fraction of samples correctly classified in the positive class among the ones classified in the positive class;
- **Recall**: $Rec = \frac{tp}{tp+fn}$; the fraction of samples correctly classified in the positive class among the ones belonging to the positive class;
- **F1 score**: $F1 = \frac{2 \cdot Pre \cdot Rec}{Pre+Rec}$; harmonic mean of the precision and recall;

3.2 Model performances

3.2.1 Scratch model

Scratch model	
Metric	Value
Accuracy	0.9046
Precision	0.8574
Recall	0.8924
F1-score(N)	0.9220
F1-score(P)	0.9383
F1-score(T)	0.7580
F1-score	0.8728

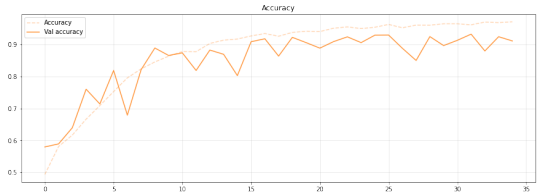


Figure 9: Scratch model train and validation accuracy

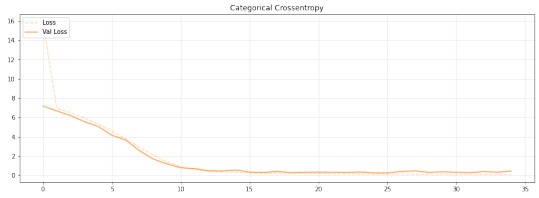


Figure 10: Scratch model train and validation loss

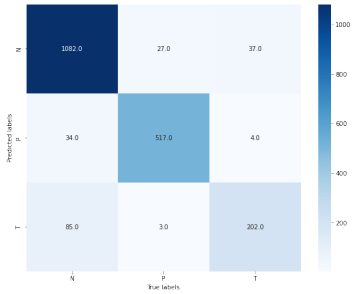


Figure 11: Scratch model predictions on the test set

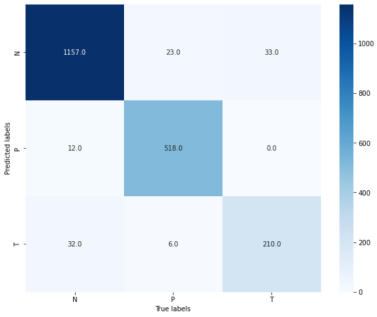


Figure 14: InceptionResNetV2 predictions on the test set

3.2.2 InceptionResNetV2 model

InceptionResNetV2	
Metric	Value
Accuracy	0.9468
Precision	0.9260
Recall	0.9248
F1-score(N)	0.9586
F1-score(P)	0.9619
F1-score(T)	0.8554
F1-score	0.9253

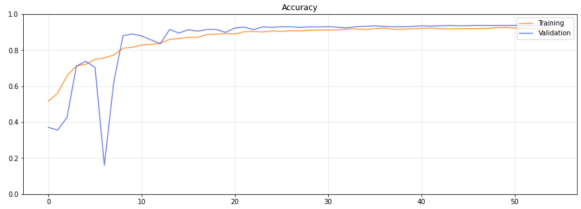


Figure 12: InceptionResNetV2 train and validation accuracy

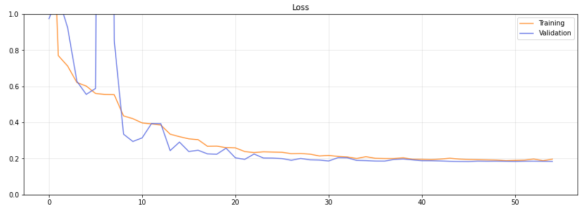


Figure 13: InceptionResNetV2 train and validation loss

3.2.3 EfficientNetV2B2 model

EfficientNetV2B2	
Metric	Value
Accuracy	0.9518
Precision	0.9319
Recall	0.9328
F1-score(N)	0.9621
F1-score(P)	0.9661
F1-score(T)	0.8689
F1-score	0.9324

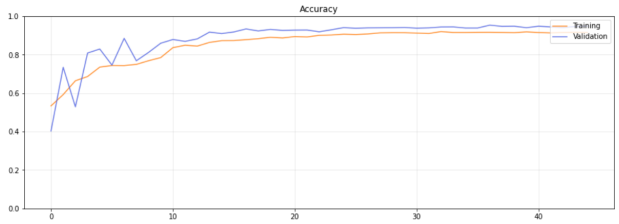


Figure 15: EfficientNetV2B2 train and validation accuracy

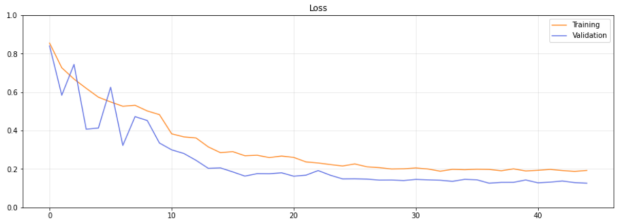


Figure 16: EfficientNetV2B2 train and validation loss

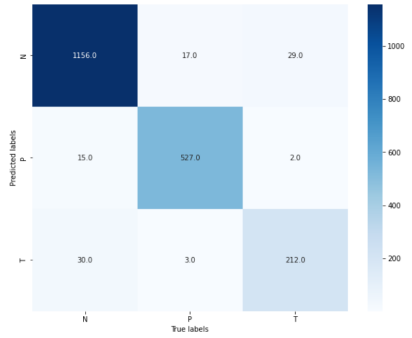


Figure 17: EfficientNetV2B2 predictions on the test set

3.2.4 Classic approach

The results obtained by simply averaging the prediction of three models are the following:

Average Ensemble	
Metric	Value
Accuracy	0.9633
Precision	0.9464
Recall	0.9470
F1-score(N)	0.9709
F1-score(P)	0.9798
F1-score(T)	0.8893
F1-score	0.9467

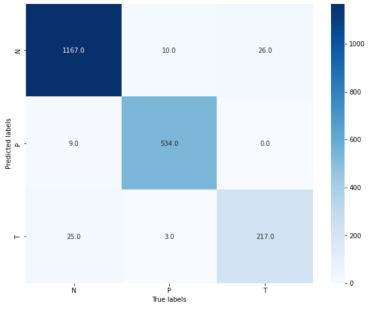


Figure 18: Average ensemble predictions on the test set

3.2.5 Confidence ensemble

Setting a confidence threshold equal to 0.99 the results for each model are reported below. The discard rate indicates how many predictions on the test set do not respect the threshold constraint.

EfficientNetV2B2	
Metric	Value
Discard rate	32.65%
Accuracy	0.9970
Precision	0.9983
Recall	0.9968
F1-score(N)	0.9974
F1-score(P)	0.9952
F1-score(T)	1.0
F1-score	0.9975

InceptionResnetV2	
Metric	Value
Discard rate	33.05%
Accuracy	0.9985
Precision	0.9991
Recall	0.9969
F1-score(N)	0.9987
F1-score(P)	0.9988
F1-score(T)	0.9966
F1-score	0.9980

Scratch model	
Metric	Value
Discard rate	32.35%
Accuracy	0.9733
Precision	0.9670
Recall	0.9554
F1-score(N)	0.9763
F1-score(P)	0.9862
F1-score(T)	0.9205
F1-score	0.9610

Instead, if we use the confidence approach on the average ensemble we can obtain even better results as reported below.

Average ensemble	
Metric	Value
Conf. threshold	0.9
Discard rate	21.95%
Accuracy	0.9949
Precision	0.9947
Recall	0.9899
F1-score(N)	0.9957
F1-score(P)	0.9967
F1-score(T)	0.9843
F1-score	0.9923

Average ensemble	
Metric	Value
Conf. threshold	0.99
Discard rate	43.90%
Accuracy	1.0
Precision	1.0
Recall	1.0
F1-score(N)	1.0
F1-score(P)	1.0
F1-score(T)	1.0
F1-score	1.0

3.3 Comparison

All the models proposed reach high performance over the test set, reaching an accuracy greater than 0.90%. Looking at the trend of accuracy and loss function of the first models it is possible to notice as the model is overfitting a little bit and it is not very stable. This is due to the fact that this model has been trained without data augmentation and without the *ReduceRLOnPlateau* callback. The second and the third models, instead, are under-fitting since the trends of loss and accuracy are identical once the model is converging and do not overfit the training dataset, but despite this they reach very high performance going over 0.95% of accuracy. For what concerns the second model the underfitting could be due to the large use of regularization techniques in the layers, while in the third model it is due to the small number of parameters.

The ensemble model, instead, combines the 3 presented models. It improves the overall accuracy up to 0.96%. This is due to fact that each model bases its prediction according to some features extracted by the network and since the models have different structures, it's like each of them looks for specific features in the data and these features are different from each model.

3.4 Important Insights

Implementing a network from scratch brought important results in terms of performance. The high performances obtained with this rather simple network can be due to the medium-low complexity of the task and the high number of images provided for the training.

Experiments showed that the validation performances when training with data split by images are better and are more optimistic than training with images split by patients. This could be due to the fact that some images belonging to the same patient and having the same class were present in both splits, causing overfitting.

Increasing the number of layers in the FC part of the network brought an increase in performance, being these layers important for the specific problem classification.

Class weights did not bring high improvement. One possibility is that the model is already performing well on the training and validation sets, and the use of class weights may not be able to improve the performance further. Having said this, even with not important improvements, handling class imbalance is theoretically necessary, and it is a procedure that should be used.

The idea of proposing two ensemble techniques is because the model can be used in two different scenarios. If the goal is to assist a doctor in his task, maybe helping him solve some doubts or showing him some area of particular interest in the images, then the classic average ensemble should be used. If instead, the goal is to almost "replace" the doctor in his task, then the second approach should be taken, or, as shown, an alternative could be to apply the concept of confidence to the classic ensemble. This last approach on the test set, setting a confidence threshold of 0.90, leads to perfect results. We haven't thought the model aims to replace a doctor considering the sensibility of the problem. In case, of course, the threshold should be increased if the doctor should not be taken into consideration for the high confidence

predictions.

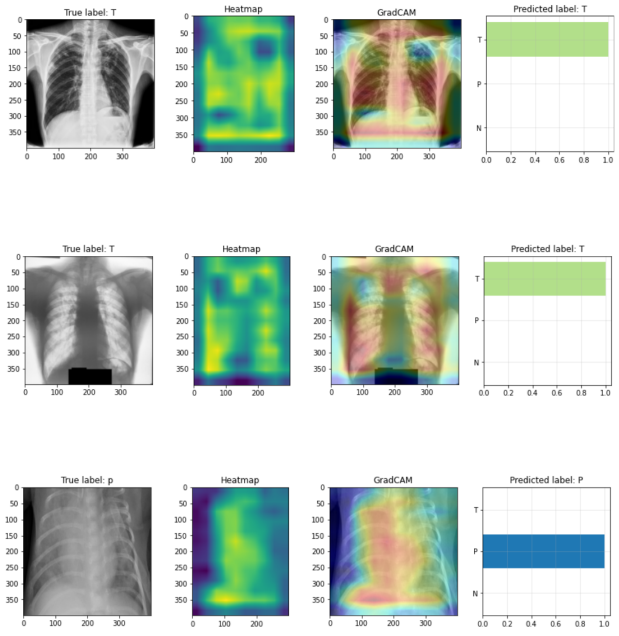
4 Discussions

4.1 Explainability

In order to have a better understanding of whether our models really "understood" where to extract features useful for the predictions we implemented 2 techniques related to the field of AI explainability: GradCam and Shapley.

4.1.1 Grad-Cam

Gradient-weighted Class Activation Mapping (Grad-CAM) [16] uses the gradients of any target concept flowing into a convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.
In our case, we computed the grad-cam with respect to one of the last convolution layers since they have the highest-level features, but in principle, it could be applied to all the layers to better understand what each filter is doing.



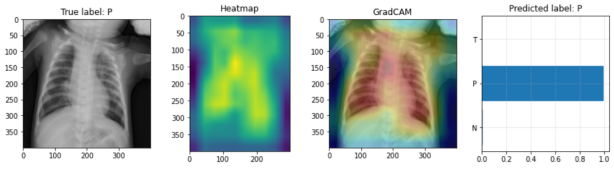


Figure 19: Example of Grad Cam results with correct prediction

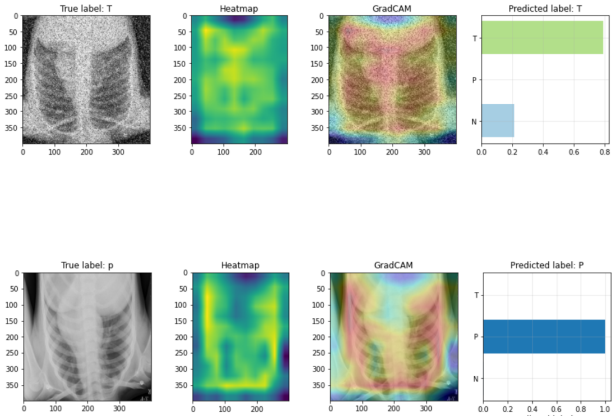


Figure 20: Example of Grad Cam results with correct prediction and images rotated by 180°

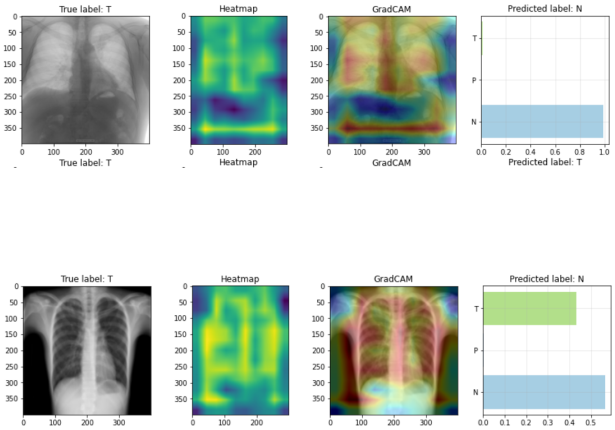


Figure 21: Example of Grad Cam results with wrong prediction

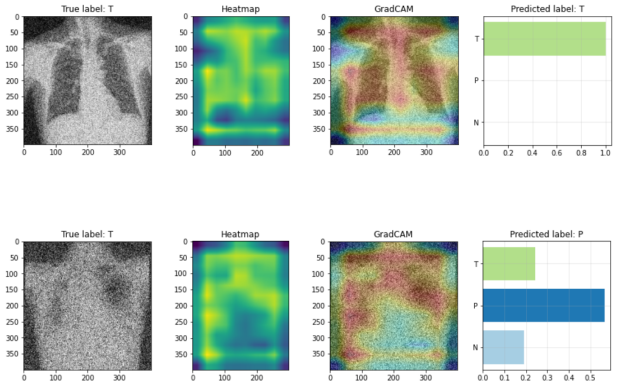


Figure 22: Example of Grad Cam results with noisy images

4.1.2 Shapley

SHAP(SHapley Additive exPlanations) [14] it's a unified framework for interpreting predictions. SHAP shows the contribution or the importance of each feature on the prediction of the model, where in this case for feature we mean a portion of the input images.

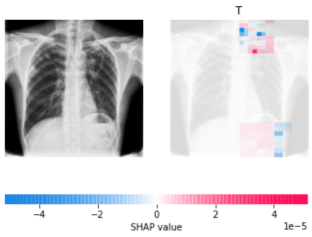


Figure 23: EfficientNetV2B2 shap values with a correct T prediction

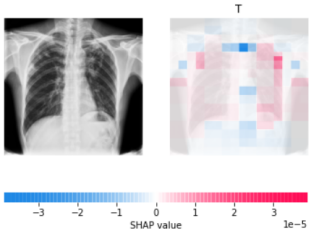


Figure 24: Inception-Resnet shap values with a correct T prediction

4.2 Comments

Looking at the results of Grad Cam it's possible to notice as generally the correct prediction are consistent, in the sense that the network is focusing on the lungs. Instead, when the prediction is wrong the focusing on the network seems to be more on the shapes of lungs and body instead than the lungs them-self, thus some predictions are biased. Also, in case of noisy images the heat-map is sparse and generally it is looking at both the lungs and the shapes, inducing wrong predictions with higher probability.

5 Future works

Possible improvements in the performance could be obtained, for example, solving the problem of the noisy images in the dataset. Because if from one point of view they help the models to be robust with respect to the noise, from an other hand we know nothing about that noise, so it's possible it destroys the information carried by the image itself (as seen before). A possibility to solve this issue has been proposed in [15] and it consists in training an encoder-decoder network that is aimed to reconstruct noisy images. An example of how train this network could be to use to use the original images plus a random noise as input, while using just the original images as target. In that case a loss function based on the reconstruction error would be used.
An other solution is to analyse the noisy image taking squared blocks of fixed size (called patches) and, for each of them, find the similar patches in the entire image as proposed in [2].
An other alternative to obtain better results could be to train other models with different structures and combine them in the *pipeline ensemble* described before: the more the models the more the number of images that get a high confidence prediction.

6 Conclusions

In this project we built three different models with different characteristics, and we proposed two different ensemble approaches, to be considered according to the usage.

The results obtained showed a high level of accuracy, proving the effectiveness of using CNNs for image classification tasks. Additionally, the use of Grad-CAM and Shapley explanation helped provide insight into the decision-making process of the model, enhancing its interpretability and reliability.

The successful classification of medical images in this project has practical implications for the medical field. By accurately identifying pneumonia and tuberculosis, the model has the potential to assist doctors in the early detection and diagnosis of these diseases. Additionally,

it could be used as a tool in telemedicine, allowing remote diagnosis and reducing the burden on healthcare facilities.

Overall, the results of this project demonstrate the potential for utilizing AI in the medical field to improve the speed and accuracy of disease diagnosis.

References

[1] IBM. What is overfitting? https://www.ibm.com/topics/overfitting.

[2] V. Katkovnik K. Dabov, A. Foi and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4271520, 2007, August 8.

[3] Keras. Callbacks api. https://keras.io/api/callbacks/.

[4] Keras. Cropping2d. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Cropping2D.

[5] Keras. Data augmentation. https://www.tensorflow.org/tutorials/images/data_augmentation.

[6] Keras. Efficientnetv2b2. https://www.tensorflow.org/api_docs/python/tf/keras/applications/efficientnet_v2/EfficientNetV2B2.

[7] Keras. Image data generator. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.

[8] Keras. Image dataset from directory. https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory.

[9] Keras. Inceptionresnetv2. https://keras.io/api/applications/inceptionresnetv2/.

[10] Keras. Randomconstrast. https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomContrast.

[11] Keras. Randomflip. https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomFlip.

[12] Keras. Randomtraslation. https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomTranslation.

[13] Keras. Randomzoom. https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomZoom.

[14] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf, 2017.

[15] G. Perrot R. Couturier and M. Salomon. Image denoising using a deep encoder-decoder network with skip connections. https://hal.science/hal-02182820/document, 2018, November 18.

[16] Abhishek Das Ramprasaath R. Selvaraju, Michael Cogswell. Grad-cam: Visual explanations from deep networks via gradient-based localization. https://arxiv.org/pdf/1610.02391.pdf.

[17] Wikipedia. Confusion matrix. https://en.wikipedia.org/wiki/Confusion_matrix.

[18] Wikipedia. Softmax function. https://en.wikipedia.org/wiki/Softmax_function.

[19] Wikipedia. Swish function. https://en.wikipedia.org/wiki/Swish_function.