Machine Learning Engineer Nanodegree

Capstone Project Report

Davide Cester
Jun 22nd, 2019

Definition

Project Overview

The medical sector is currently seeing widespread research efforts in the field of diagnosis automation; the potential benefits are huge both in terms of cost effectiveness and quality of treatment. This project demonstrates the use of Machine Learning techniques to automate the analysis of X-Ray scans.

The possibility of using ML methods for medical diagnoses has been already demonstrated in literature and in some cases the algorithms have even shown better performances compared to human diagnosis (see [1] and [2] for examples). One of the most popular ML tools in the field of image analysis, and more generally for pattern recognition problems, are Convolutional Neural Networks (CNNs); these networks can be developed from scratch, or starting from some pre-trained network that is already able to recognize basic features like shapes, edges or colors.

In this project I made use of the Kaggle dataset linked in [3]; the dataset contains several Chest X-Ray Images of pediatric patients and is one of the datasets used in [2], which provides a convenient reference for the project performance. The model used here is based on the pre-trained MobileNet (see [4], [5]) with the addition of custom layers on top.

Problem Statement

The project focuses on the task of correctly diagnose pneumonia from standardized chest X-Ray images, based on the fact that pneumonia causes anomalies in the images when present. The analysis will return a boolean output indicating if the patient has been diagnosed or not.

More in detail, the tasks composing the solution are as follows:

1. pre-process the images to enforce uniform characteristics (size, orientation, color mode)
2. analyze the category distribution of the dataset and rearrange it if required
3. build and optimize a CNN model based on a pre-trained networks
4. assess the model performance and discuss the results

Metrics

A common metric for binary classifiers is Accuracy, which is defined as the ratio of all correct answers over all samples: $A = (TP + TN) / (TP + FP + TN + FN)$. An accuracy value of the model will be reported for the training dataset.

However simple and well-defined, accuracy is not always an useful metric, particularly when dealing with medical-related applications. When assessing a diagnosis algorithm, the possibility of sending home a sick patient should be considered separately from the possibility of doing an extra exam on a healthy patient; accuracy weights equally all errors and does not describe with sufficient detail the model performance.

When the diagnosis has a binary output (e.g. sick/healthy) like in this project, ROC curves provide a better metric. ROC (Receiving Operating Characteristic) curves are diagrams of the True Positive Rate versus the False Positive Rate, when the discrimination parameter is changed. In addition, the AUC parameter (Area Under the Curve) provides another useful estimate: the closer the AUC is to 1.0, the smaller is the compromise the algorithm has to make between True Positives and False Positives[1].

Analysis

Data Exploration

The official description of the dataset [3] states:

> *"The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are 5,863 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).*
>
> *Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.*
>
> *For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert."*

The images are in grayscale and all of good quality, however the size and aspect ratio can vary significantly. Moreover, the two "normal" and "pneumonia" populations are unbalanced, and the validation subset is definitely too small. These issues will be addressed in the preprocessing stage, except for the normal/pneumonia unbalance.

	Training	Validation	Test
Normal	1341	8	234
Pneumonia	3875	8	390

Table 1: distribution of images in the original dataset

1 There is another type of ROC curve used in medical application, where the plot displays Specificity versus Sensitivity. However this project uses the standard ROC curve as metric for better comparison with [2].

Exploratory Visualization

The following two images clearly demonstrate the different sizes and aspect ratios one can find in the dataset. It is also worth noting that some images are not completely centered, show some rotation of the subject or feature different contrast levels. Rather than software compensate these features, they are used as a sort of built-in data augmentation.

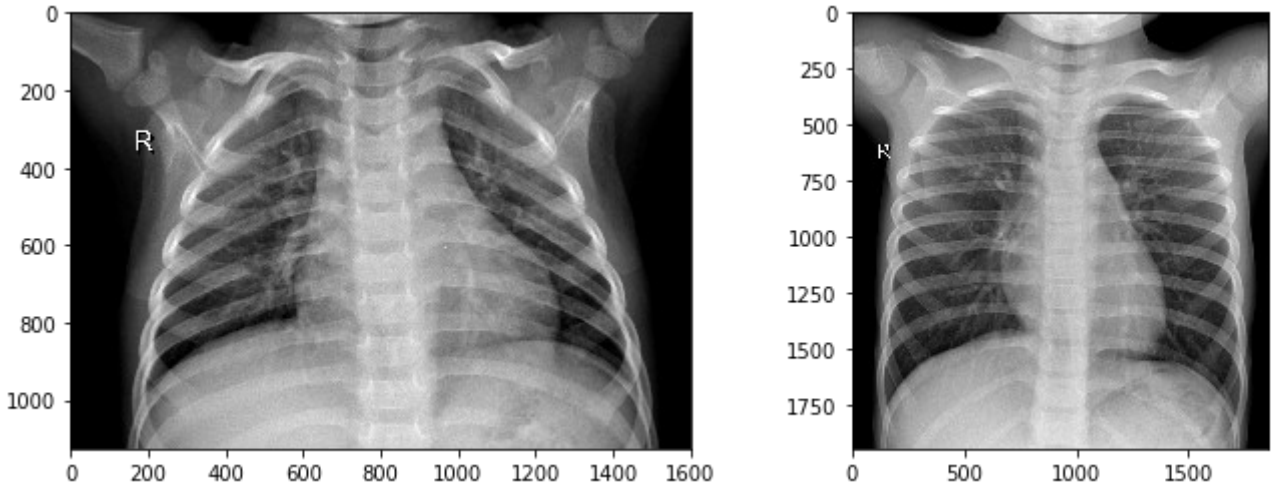

Fig. 1: examples of raw images from the dataset

Algorithms and Techniques

To analyze the X-Ray images this project uses a Convolutional Neural Network (CNN), which belongs to the family of Deep Learning algorithms and is particularly well-suited to solve image analysis problems.

In case of image categorization problems a CNN outputs a set of numbers, each one representing the probability that the input image belongs to the corresponding category. Although special settings are available for the case when there are just two categories (binary classification) so that the CNN directly emits the category identifier, I did not use them in order to have access to all the individual probabilities and to be able to quickly attach the ROC calculations to the analysis pipeline.

The building blocks of a CNN are called layers and are stacked in a sequence, each one corresponding to a particular stage of data manipulation. Generally speaking, one could say that the layers close to the input detect basic features such as edges or corners, while the layers close to the output are sensitive to more advanced features like shapes. It is a matter of fact that the low-level layers of a trained CNN can be successfully reused without modifications in a different CNN, provided that the input data has at least some basic similarities (e.g. they are all RGB images of a certain size). This method is called *Transfer Learning*, and when it is applied only the higher levels of the CNN (usually a fraction of the total) have to be trained, with great saving in time and computational resources. This project is build on the MobileNet pre-trained CNN available in Keras (see [5]) with the addition of a few custom layers.

The CNN performance can be optimized by tuning the following parameters:

- the number and type of the layers (the *architecture* of the CNN);
- the optimizer, loss function and metric functions: these parameters control the inner mathematics of the learning process;

- the number of epochs, or the number of learning iterations;
- the batch size for the image *generators*: the generators allow to load into memory just a fraction of the images at the time, while still processing all of them.

The CNN was trained on a personal computer with a GPU; this posed some resource constraints on the choice of the architecture and of the parameters, mainly regarding RAM usage.

Benchmark

For the same problem/dataset and a similar solution, reference [2] provides a ROC curve to summarize the performance of the model (Fig. 6.E, here reported as Fig. 2); this project is evaluated by comparing the ROC curve and the underlying area (AUC).

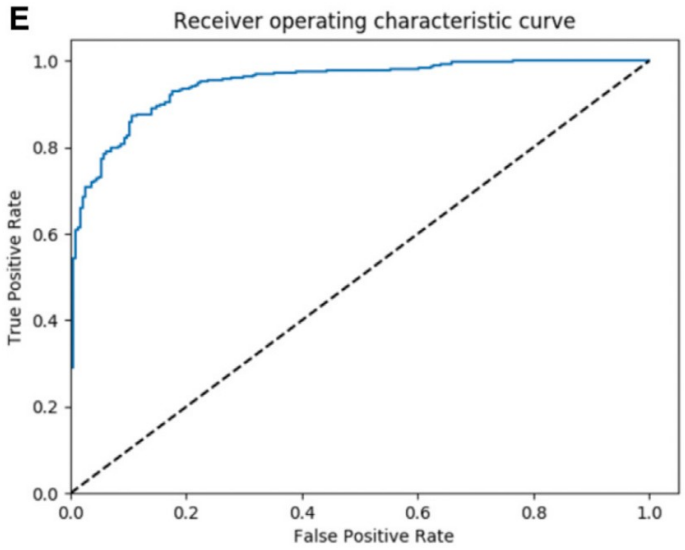

Fig. 2: ROC curve for the same dataset used here, AUC = 96.8% (from [2])

Methodology

Data Preprocessing

All the images must be converted to a size and a format suitable for being used as input to MobileNet pre-trained network layers, namely 224x224 pixels in RGB mode.

The aspect ratio of the images is not constant through the dataset and the built-in resize functions in Keras stretch the images until they completely match the destination size. This default behavior would have produced randomly distorted and non-natural chest images; therefore I decided to preserve the correct aspect ratio of each image by writing a custom scaling procedure:

1. a temporary grayscale image with the proper size is created
2. the temporary image is filled with random noise
3. the temporary image is extended to RGB color mode to match the CNN input
4. the X-Ray image is scaled to fit the final 224x224 square but preserving the aspect ratio (the imaged is scaled down until both width and height are lower than or equal to 224)
5. the scaled image is copied and pasted onto the noisy image, in a centered position

The resulting image has the proper characteristics to match the MobileNet input stage, while at the same time preserving physical information about the patient without any distortion. The random noise was chosen to avoid introducing any feature into the resized images; due to different aspect ratios of the original images, it will appear as either two vertical bands or two horizontal bands around the X-Ray scaled image (see Fig. 3 as an example).

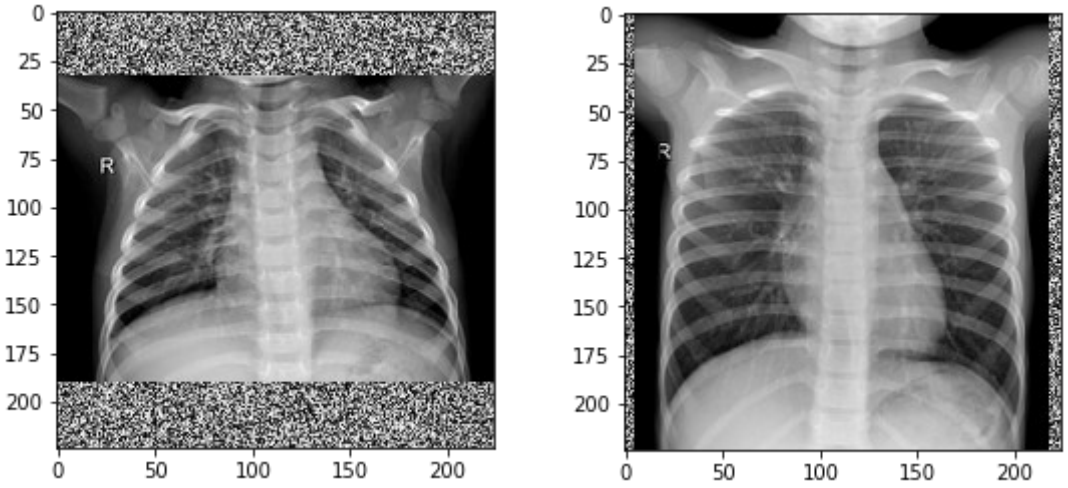

Fig. 3: the same images from Fig. 1 after preprocessing

In addition, to address the issue of the lack of statistics in the validation subset, the validation and test subsets were merged with the training subset and recreated as a fixed percentage of the full dataset (with the ratios 60:20:20).

	Training	Validation	Test
Normal	949	317	317
Pneumonia	2563	855	855

Table 2: distribution of images in the preprocessed dataset

The rearranged dataset still suffers from normal / pneumonia unbalance like the original one; however this unbalance is now constant across the three subsets and can be measured and compensated with the proper tools.

Implementation

The CNN used in this work is based on the pre-trained MobileNet model available in Keras. The MobileNet model is imported without its top layer and with parameters pre-trained on the ImageNet dataset [6].

The following custom CNN layers have been added on top of the MobileNet:

1. a Dropout layer: this will turn off a random fraction of the “neurons” at every learning step and helps reducing overfitting;
2. a GlobalAveragePooling2D layer: averaging layers reduce data dimensionality, helping in getting better training speed and preventing overfitting;

3. a Dense (fully connected) layer: this is a normal layer with learning “neurons”, their number being a parameter of the layer;
4. another Dropout layer;
5. the final Dense layer with output size equal to the number of categories.

The number and type of the custom layers is based on a previous project for dog breed recognition, with minor adjustments. The final configuration of the network, including the size of Dense layers and the reduction parameter of the Dropout layers, has been experimentally determined, using the following criteria to determine if a configuration was an improvement:

- the learning speed is significantly faster and performance does not decrease heavily, or
- the final performance in terms of ROC curve and AUC shows significant improvement.

The following schema is a summary of the final layout of the CNN:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenet_1.00_224 (Model)   (None, None, None, 1024)  3228864
_________________________________________________________________
dropout (Dropout)            (None, None, None, 1024)  0
_________________________________________________________________
global_average_pooling2d (Gl (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 130
=================================================================
Total params: 3,294,594
Trainable params: 65,730
Non-trainable params: 3,228,864
_________________________________________________________________
```

One issue that happened at an early stage of development was a ResourceExhaustionError at the moment of loading the data. The code taken from a previous, smaller project was replaced with the use of Keras ImageDataGenerator() objects; they allow to load images in batches. The size of the batches was also optimized to take into account the memory limitations on the local GPU. Although the DataGenerators could also do resizing and other preprocessing on the images, I decided perform these operations in a complete separate block of code at an earlier stage, trading processing time for disk space (which is available in high quantities).

The model was compiled with the following parameters:

- *adam* as optimizer
- *categorical crossentropy* as loss function
- *accuracy* as metric function

Refinement

Once the final architecture was finalized, the use of Data augmentation methods was also considered. Data augmentation is a powerful preprocessing strategy that allows to introduce

additional variability in the input data not present in the original dataset, like offsets, rotations, color alterations and so on. It usually helps improving performances, e.g. by removing undesired and hidden correlations between features and spatial position or orientation.

The following augmentations have been attempted by setting the corresponding parameters of the data generators:

- vertical shift in the range ±10 pixels
- horizontal shift in the range ±10 pixels
- rotation in the range ±20 degrees
- brightness change in the range ±20%

These parameters have been tested in different combinations, e.g. only rotation and brightness.

Contrarily to expectations, at the end of the optimization stage no relevant performance improvement was observed associated to the use of data augmentation. With the mindset of minimizing the processing effort unless it provides tangible benefits, data augmentation was completely excluded from the final model.

My explanation for this behavior is that the original dataset was clearly composed of images from different setups and with different settings, so that it already contained the amount of variability that data augmentation would have introduced in a more uniform dataset (see Fig. 1 in section “Exploratory Visualization” for a visual example).

Results

Model Evaluation and Validation

The model was trained using a validation set as training reference. Fig. 4a and 4b show the evolution of model metrics across learning epochs.

Both plots show good convergence towards high accuracy and low loss; at the same time there is good agreement between the train and the validation dataset, which is a good indicator that the model is not overfitting. The trend of convergence also suggests that probably the optimal number of training epochs for this architecture is around 10.

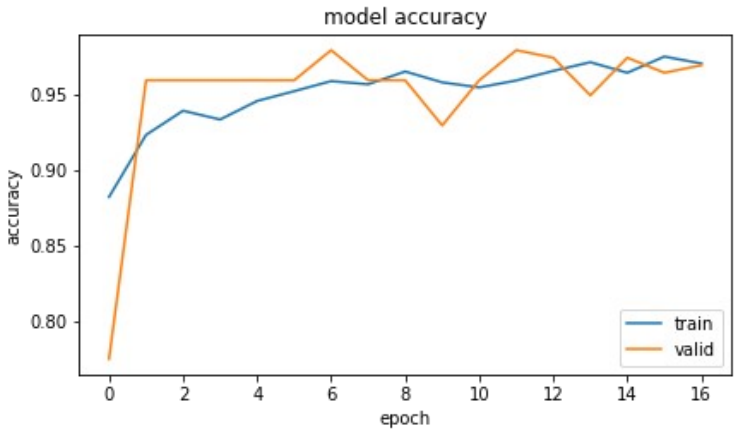

Fig. 4a: evolution of model accuracy across learning epochs

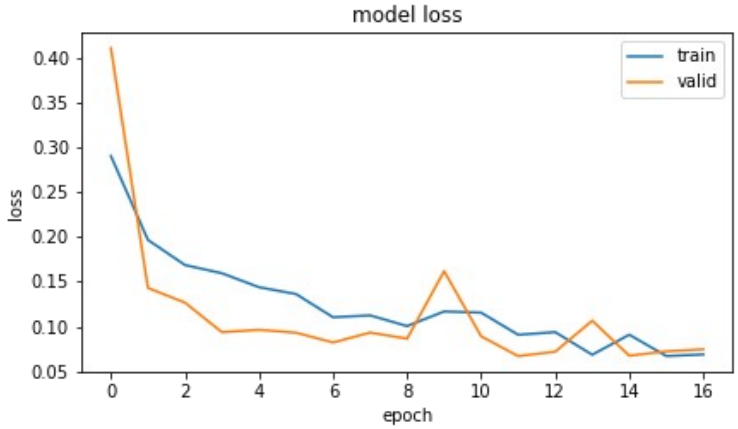

Fig. 4b: evolution of model loss across learning epochs

Justification

At the end of the training the ROC curve and the AUC parameter were calculated as benchmark; they are reported in Fig. 5.

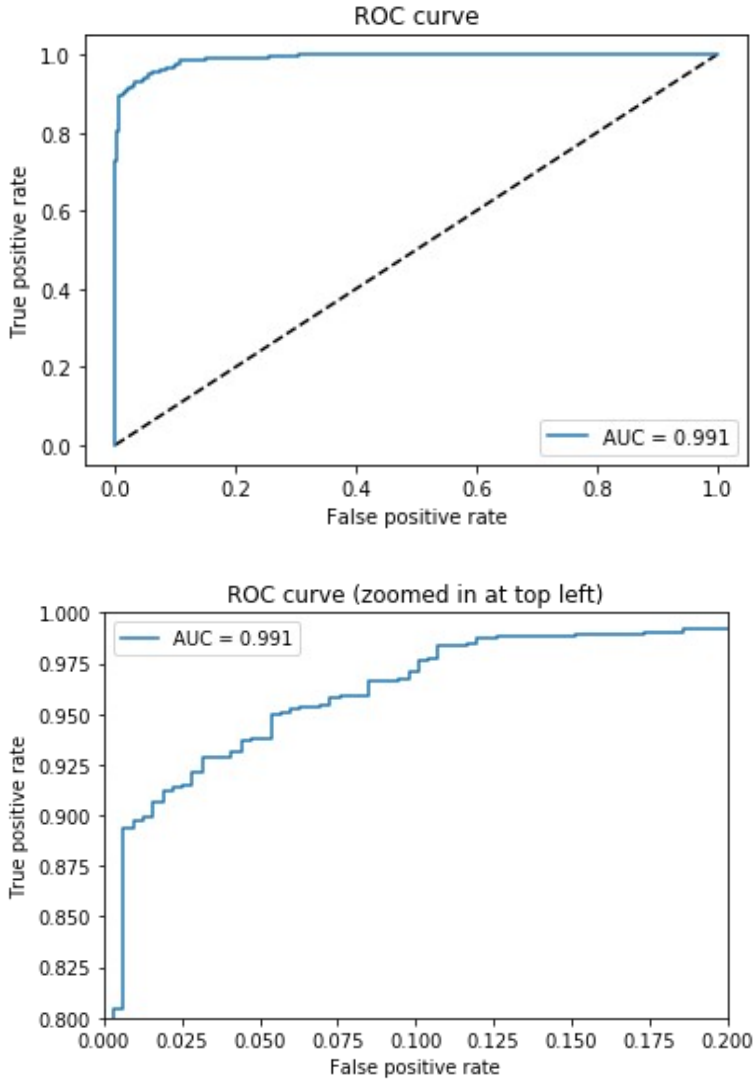

Fig. 5: ROC curve and AUC for the final CNN architecture

The reference performance from [2] was previously reported as Fig. 2, while a superimposition of the two curves is shown in Fig. 6 in the next section. The main benchmark value for a ROC curve is the value of the underlying area, or AOC; the AOC value for this project is 0.991, to be compared with 0.968 of the reference article, a result that can be considered satisfactory.

The Confusion Matrix is a table that provides additional detail about the performance of a classification algorithm in terms of the actual classes versus the predicted classes; due to different conventions on axis labeling, I reported in Tab. 3 the structure of a Confusion Matrix as defined in Keras[2].

Input \ Output	Negative	Positive
Normal	True Negatives	False Positives
Pneumonia	False Negatives	True Positives

Table 3: generic structure of a confusion matrix in Keras

Tab. 4 reports the actual values referred to the test subset.

Input \ Output	Negative	Positive
Normal	293	24
Pneumonia	38	817

Table 4: confusion matrix for the final run of the model

From the values in the Confusion Matrix, additional indicator values can be calculated. When dealing with medical applications an important indicator is Recall, defined as the fraction of Positive samples that have been correctly classified, or in numerical terms:

$$Recall = \frac{TP}{TP + FN}$$

The values reported in Tab. 4 return a Recall value of 95.56%, meaning that for 4.44% of the hypothetical sick patients this algorithm would recommend their dismissal without treatment.

Conclusion

Free-Form Visualization

Fig. 6 shows a comprehensive plot that summarizes performance of both this project's model and the reference one. The best possible ROC curve features a vertical line at FPR = 0 and an horizontal line at TPR = 1 making a right angle at the top-left corner; such a curve would have an AUC of 1.0.

The model developed in this project clearly returns better results than the reference, with an AUC of 0.991 compared to 0.968.

2 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

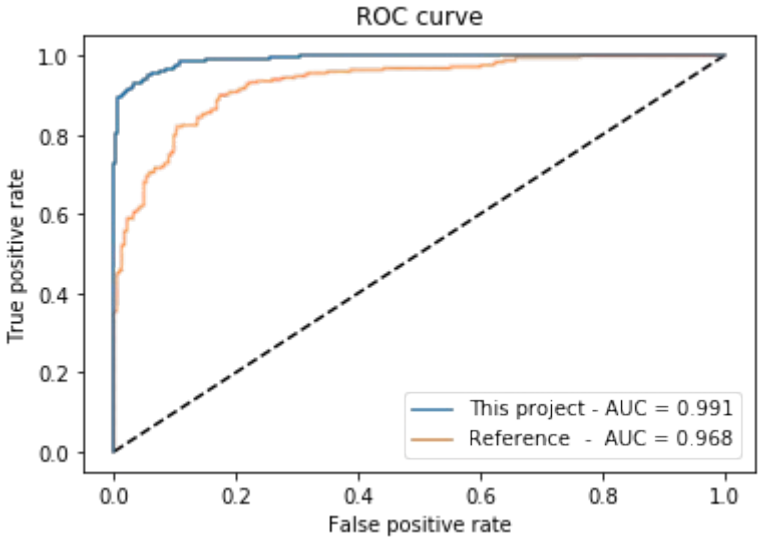

Fig. 6: superimposition of ROC curves for this project and the literature reference

Moreover, recalling that True Positive Rate is a synonym for Recall, and that for a binary classifier the last operation is always a simple threshold filter, it is interesting to estimate what would be the lowest FPR corresponding to a TPR of 1.0. This question is equivalent of asking, how many healthy patients would this algorithm diagnose with pneumonia, if we change the threshold until we are sure to catch all sick patients, which is the optimal goal when diagnosing life-threatening diseases. This question is also an important benchmark for real application, because each incorrect diagnose on a healthy patient represents a cost for both the individual patient and the whole health system.

From Fig. 6 a rough estimate returns that in order to correctly classify all the pneumonia scans, one is forced to also classify as sick 65% of healthy patients in case of the reference model, and only 30% in case of this model, and this is also a good result.

Reflection

This project was developed using the following steps:

1. finding a dataset with an interesting and concrete associated application
2. inspecting the dataset and preprocessing the images
3. developing a CNN model, initially with a trial-and-error approach, and then grid search
4. training the model (multiple times)
5. calculating the benchmarks

I believe one of the most interesting part was the search and exploration of available datasets, because I had the opportunity to think about real life problems that could be solved with Machine Learning, and I also had to imagine how I could do that in order to select one of them for the project.

The trickiest part was definitely the image preprocessing, because there are many parameters that one has to put under control before the data become really usable. As an example, the image generators available in Keras have a built-in resize function and the model seems to run smoothly without explicitly dealing about image preprocessing; however the final performance turns out to be quite bad and difficult to improve. Also, for some reason the original dataset provides a very small

validation subset, which made the learning process very unstable and prompted me to redistribute the images in the three folders. In the end I got convinced that data preparation accounts for at least 50% of a project, and even the best presented dataset should be throughly inspected by a Machine Learning Engineer to make sure it really fits the project.

Improvement

From a dataset point of view, I believe the next step in developing an automated diagnosis software would be to extend the dataset with images from non-pediatric patients, to make it more general. Also, increasing the absolute number of available images would definitely not hurt.

As for the algorithm, there are for sure different model architectures that provide some better results. Having time and computational resources, one could further search for optimizing the number and type of layers; however this process might go on indefinitely, and one should always look for a compromise in terms of resources and performance. In this project the development of the classifier was considered completed when the model showed significant better performance compared to the designated reference.

Should the algorithm be developed to be embedded in a medical device or similar application, I would definitely redesign the image preprocessing stage. In the current implementation the images are preprocessed at an earlier stage and cached on disk in a second folder hierarchy; this has the advantage to speed up the training but is not optimized for usage with an on-line classifier, where the model is pre-trained and there is just one image in memory coming from the sensors. In such a case I would write a dedicated pre-processing function that can be used as callback by Keras image handlers and does the same work at the moment of prediction.

References

[1] “Dermatologist-level classification of skin cancer with deep neural networks” doi:10.1038/nature21056

[2] “Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning” doi:10.1016/j.cell.2018.02.010

[3] https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

[4] “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications” https://arxiv.org/abs/1704.04861

[5] MobileNet documentation https://keras.io/applications/#mobilenet

[6] ImageNet dataset official website: http://www.image-net.org/