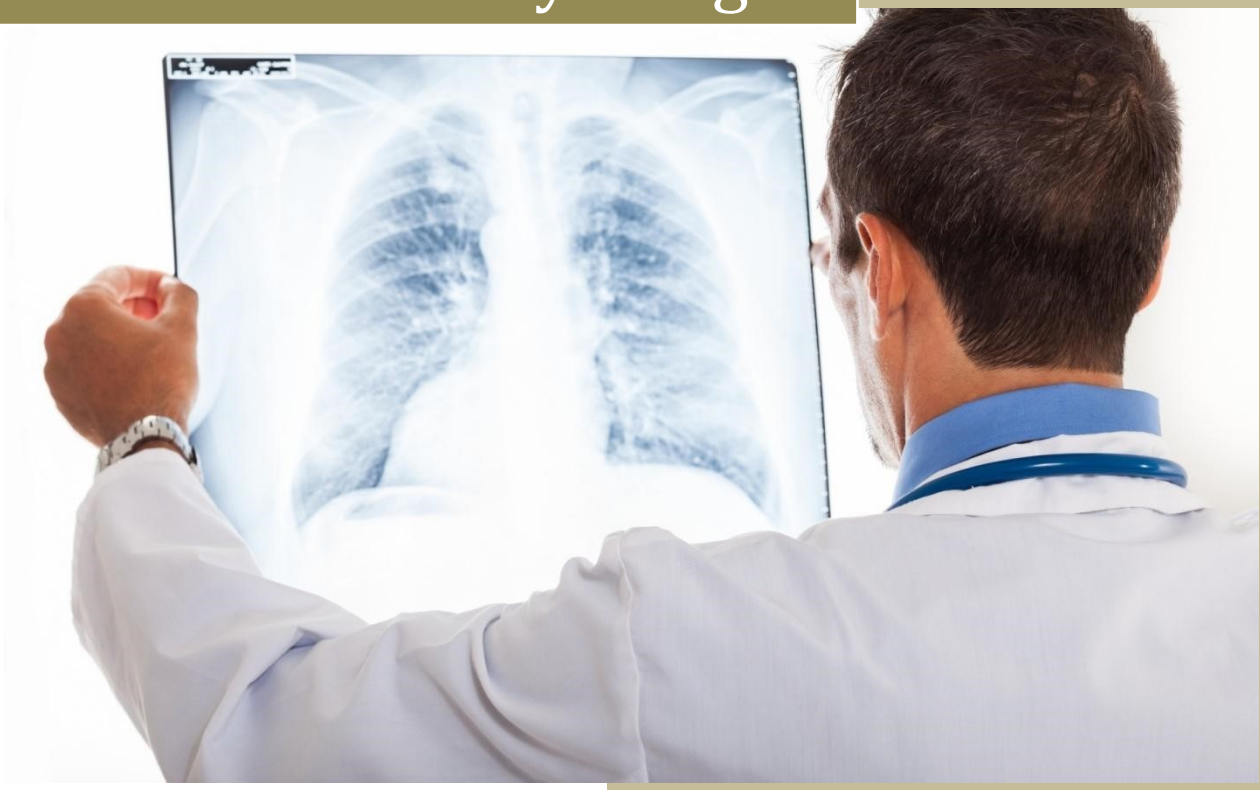
Machine Learning Engineer Nanodegree

Detecting Children Pneumonia From Chest X-Ray Images



Beáta Hargitay

1/30/2019

I. Definition

Project Overview

Domain Background

Nowadays the usage of Clinical Decision Support Systems (CDSS) are spreading like a virus and being used more and more in healthcare institutes. But what is a CDSS? It is a system that helps in clinical decision-making and diagnosing for health professional with grounded data analysis based on other patients with similar symptoms. Using CDSS not only can help to improve the quality of healthcare but can also drastically reduce costs. These systems' history takes back until the early 1970s and now is major topic in Artificial Intelligence.

My personal motivation for investigating this domain is that, I think CDSS is more than interesting and has a lot of undiscovered subdomains that would be great for the entire humanity to have. Maybe one day, Clinical Decision Support Systems are going to be advanced enough to detect cancer in an early stage, saving millions of lives.

To this day there are several academic works related to Clinical Decision Support Systems, and being published quite frequently. Later in this paper I am going to compare my work with one of Stanford Machine Learning Group's work.

Problem Statement

Pneumonia is the single largest infectious cause of death of children worldwide. This disease is accounting for 16% of all deaths under the age of five (which meant 920 136 children in 2015). Pneumonia is most prevalent in South Asia and sub-Saharan Africa but it is present all over the world. It can be treated with low-cost, low-tech medication and care if it is diagnosed in time. (World Health Organization, 2016)

My project's aim is to build a base for a CDSS for radiologists to help diagnose Pneumonia easier, quicker and more accurately from children's chest X-Ray images. My intention is to create a model that might be implemented into a software or a webpage through an API.

Metrics

All the metrics that I used for evaluating my models are performance measurements for machine learning classification problems with at least two classes. I chose these metrics because the dataset has unbalanced number of images for the two classes, Normal and Pneumonia (as confirmed later in the 'Analysis' chapter). I would like to state that not all of the listed metrics were used at the evaluation step but still demonstrating them is essential to understand all the chosen metrics.

Confusion Matrix

The confusion matrix is a table with the combinations of actual and predicted values. It shows the actual (real) values in its columns and the predicted values in its rows *(Figure 1).* Thanks to this matrix we can precisely identify the number of cases that are predicted well, and the ones that are not.

To gain more understanding of the confusion matrix, we should understand the following concepts:

True positive (TP): the model predicted yes and it is true
True negative (TN): the model predicted no and it is true
False Positive (FP): the model predicted yes and it is false
False negative (FN): the model predicted no and it is false

Confusion matrix is extremely useful for measuring Recall, Precision and AUC-ROC Curve.

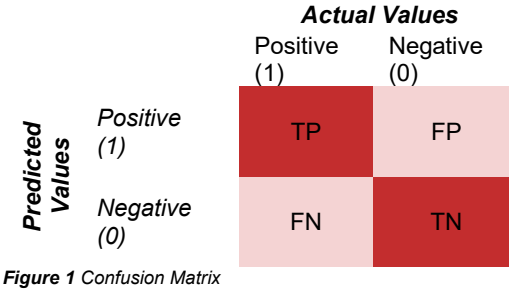


Figure 1 *Confusion Matrix*

Precision

The precision tells us, how good our model predict in all the classes (Negative and Positive as well), i.e. how good we could predict in which group does somebody belongs to. This value should be as high as possible as in this case we would only want to give treatments to the patients who are really ill.

$$Precison = \frac{\sum True\ Positive}{\sum True\ Positive + \sum False\ Positive}$$

Recall

The recall tells us, how good did the model predict the positive cases, i.e. how good could we predict if somebody has Pneumonia.

$$Recall = \frac{\sum True\ Positive}{\sum True\ Positive + \sum False\ Negative}$$

Specificity

The specificity tells us, how good did we predict the negative cases, i.e. how good could we predict if somebody is healthy.

$$Specificity = \frac{\sum True\ Negative}{\sum True\ Negative + \sum False\ Negative}$$

F1 Score

We use F1 Score to seek a balance between Precision and Recall with an uneven class distribution. This was animportant metric for me as the dataset had more images in the ‘Pneumonia’ class (which will be demonstrated in the ‘Analysis’ chapter).

$$F1\ Score = 2 * \frac{Precision\ *\ Recall}{Precision + Recall}$$

ROC and AUC

The **ROC** (Receiver Operating Curve) shows the trade-off between recall and specificity. It plots recall on the y-axis against specificity on the x-axis by varying the threshold *(Figure 2).* This plot can tell us how good the model can distinguish between 2 classes (in our case if a patients has Pneumonia or not). The dotted diagonal line corresponds to a classifier with random chance. The closer the ROC curve gets to the upper left corner, the better the classifier is.

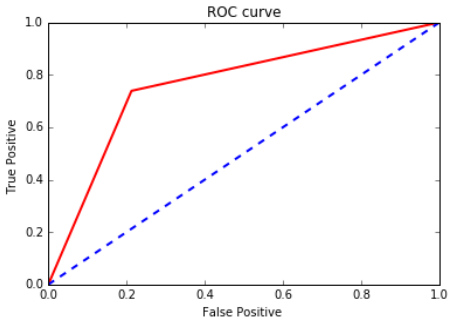


Figure 2 *ROC curve*
Source: http://enhancedatascience.com/2017/04/23/tutorial-logistic-regression-python/

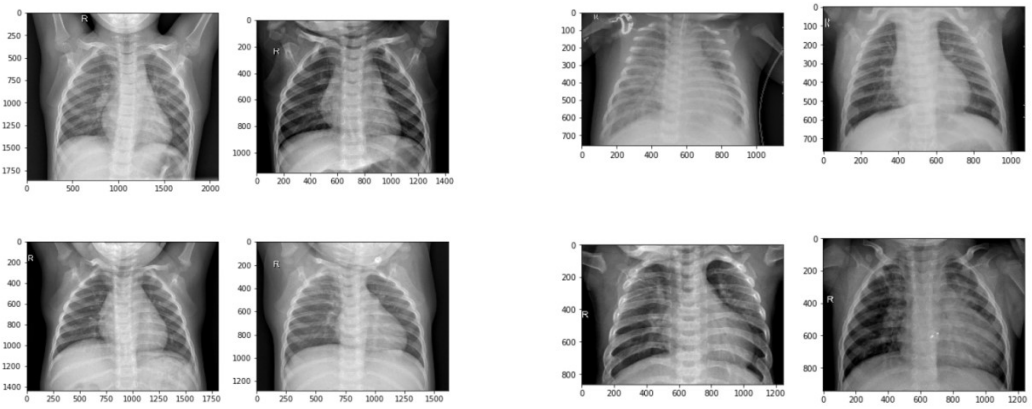
The ROC curve is a valuable graph, but by itself does not constitute a single measure of performance of a classifier. It can be however used to produce the **AUC** (Area Underneath the Curve) metric which is the total area under the ROC curve. An AUC of 1.0 indicates a perfect classifier, which means that the model's prediction is 100% right and an AUC of 0.5 indicates a completely ineffective classifier (the diagonal line) which means that the model could not get better results than the random chance. (Bruce & Bruce, 2017) We can claim of a classifier that it is working well, if it has an AUC value above 0.7.

During the evaluation step I used the confusion matrix, the Precision, the Recall, the F1 Score and the AUC value to determine the goodness of the given model on the test set.

II. Analysis

Data Exploration

The dataset used for this project were downloaded from Kaggle. It contains 5 863 JPG files organized into 3 folders (train, test, val for training, testing and validation respectively) which all have 2 subfolders for each image category: Pneumonia and Normal *(Figures 3 and 4).*



Figures 3 and 4 *Examples on X-ray images with no Pneumonia (left) and Pneumonia (right)*

“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.” (Mooney, 2018)

Exploratory Visualization

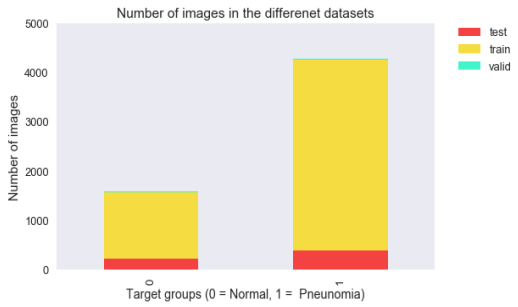
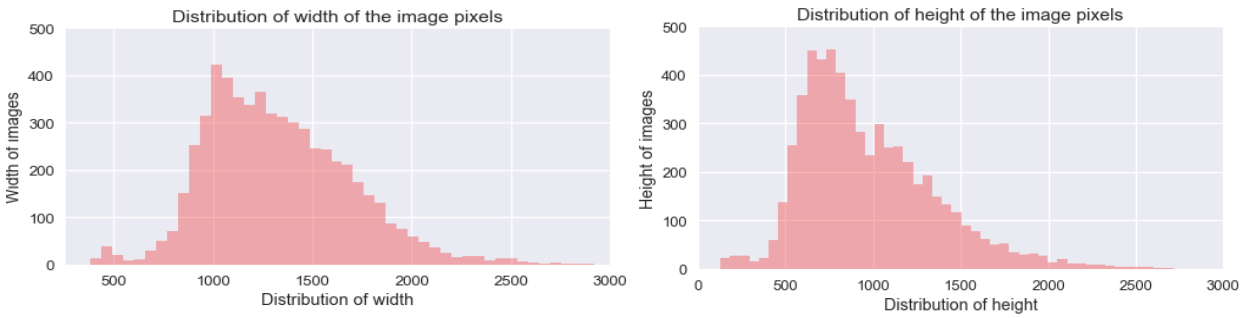


Figure 5 *Density of different datasets according to the target variable*

The dataset contains 3 subsets a train, a test and a validation set. Each set has samples from each cases of the target variable: Normal (0) and Pneumonia (1). As we can see on *Figure 5* the sizes of the subsets and the density of the target variable is unbalanced. As usual, the training set is much bigger than the others, but what looks interesting at first glance is that the validation set is so small that it almost cannot be even seen on the figure. I believe that a small validation set like this one (with only 16 images) cannot give a reliable result during the evaluation of a performance of the model. This why I used the test set for validation during the training process as well.



Figures 6 and 7 *Density of image height (left) and width (right)*

I have also analyzed the distribution of pixel size of the images belonging to the different sets. *Figures 6 and 7* show the density of image heights and widths respectively. We can see at the first glance that the density of image pixels differs a lot from each other.

Algorithms and Techniques

My solution for a medical diagnostic tool that could recognize Children Pneumonia from chest X-ray images is a Deep Learning Model built with Keras with Tensorflow backend. Since the dataset I am going to use has a small number of images in the training set (only 5 216) it would make no sense to create my own model from scratch. This is why I used exclusively transfer learning for solving this problem.

Transfer learning is a popular method in computer vision as it allows us to build quite accurate models quick and easy even with very little data. With this technique we can use a model which is already trained

on a large benchmark dataset (usually on ImageNet[1]) and repurpose it to our own needs. This means that instead of starting the model from scratch, we can use one is that already built up and learnt how to recognize patterns and edges when solving a whole other problem. To use these predefined models to predict the classes of our specific dataset, we simply have to add a new classifier at the end of that specific network. This way we can use the pretrained features to recognize completely new objects or patterns that were not seen in the benchmark training at all.

It is a common practice to use models from published literature (e.g. Inception, Resnet, VGG, Mobilenet) and luckily some of them are implemented into Keras as well *(Figure 8)*. When choosing the architectures which I wanted to use for modelling, my criterion was to select the 3 architectures that perform best in the top 1% on the ImageNet dataset. We can see on *Figure 8* the models in sequence according to their accuracy in the top 1%. According to this table the very best architecture is NASNetLarge. I chose to exclude this architecture as it was designed to build up a model architecture in real-time directly on the dataset of interest so it would have taken up too much resources to run. This is why I ended up choosing the Xception, InceptionResNetV2 and the InceptionV3 architectures. They all have Top-1 Accuracy above 0.77 and Top5-Accuracy above 0.93.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
Xception	88 MB	0.790	0.945	22,910,480	126
InceptionV3	92 MB	0.779	0.937	23,851,784	159
DenseNet201	80 MB	0.773	0.936	20,242,984	201
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet121	33 MB	0.750	0.923	8,062,504	121
ResNet50	99 MB	0.749	0.921	25,636,712	168
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
MobileNet	16 MB	0.704	0.895	4,253,864	88

Figure 8 *Implemented models in Keras (basedon their Top-1 accuracy) in descending order*
Source: https://keras.io/applications/

Benchmark

As a benchmark model I used Stanford Machine Learning Group's ChestXNet model. The ChestXNet is a deep learning algorithm which can detect Pneumonia from chest X-ray images at a level exceeding practicing radiologists. They also extended CheXNet to detect all 14 diseases in ChestX-ray14 database and achieved state of the art results on all 14 diseases. (Rajpurkar, and his colleagues., 2017) As we can see on *Figure 9* the ChestXNet has a performance of 0.7680 AUC on Pneumonia prediction. My goal is to get even better results on my test set.

Pathology	Wang et al. (2017)	Yao et al. (2017)	CheXNet (ours)
Atelectasis	0.716	0.772	**0.8094**
Cardiomegaly	0.807	0.904	**0.9248**
Effusion	0.784	0.859	**0.8638**
Infiltration	0.609	0.695	**0.7345**
Mass	0.706	0.792	**0.8676**
Nodule	0.671	0.717	**0.7802**
Pneumonia	0.633	0.713	**0.7680**
Pneumothorax	0.806	0.841	**0.8887**
Consolidation	0.708	0.788	**0.7901**
Edema	0.835	0.882	**0.8878**
Emphysema	0.815	0.829	**0.9371**
Fibrosis	0.769	0.767	**0.8047**
Pleural Thickening	0.708	0.765	**0.8062**
Hernia	0.767	0.914	**0.9164**

Figure 9 *ChestXNet's Performance (in AUC) on all 14 pathologies in the ChestX-ray14 dataset. (Source: Rajpurkar, and his colleagues., 2017)*

[1] **ImageNet dataset:** ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images. Currently there is an average of over five hundred images per node. *Source:* http://www.image-net.org

III. Methodology

Data Preprocessing

The data processing consisted of 3 main parts: the image coloring, the resizing and rescaling and the image augmentation part. All of these steps are very important as they help our model to avoid overfitting.

Image coloring

For me as a layman, looking at the X-ray images tells nothing about Pneumonia. This is why I decided to put some colormap on the images, to help me (and also the computer) to find the differences easier. I tried out a lot of different colormaps but I chose 'terrain' *(Figure 10)* at the end because it emphasized the differences the best.

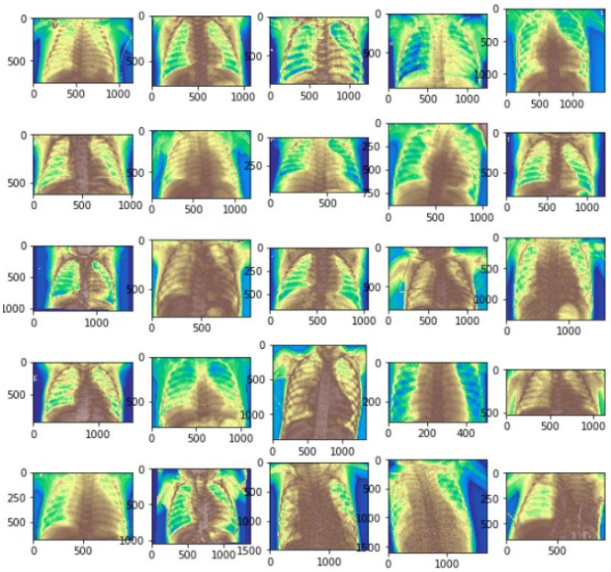


Figure 10 *''Terrain" colormap applied on some random images from the train set*

Resizing, rescaling

To train our model to identify something, we must use a fixed size (width and height) for the images that we use as input, i.e. we must resize them. To find the most reasonable image magnitude, I looked at the statistics of image pixels on *Figures 6* and *7* and also researched a little in the documentations of the chosen model architectures. This is how I ended up choosing the size of 150*150 pixels for the images.

Rescaling our images is also important because standardizing the values of the image pixels (from 1 to 255 to 0 to 1) helps our neural network to perform better.

Image Augmentation

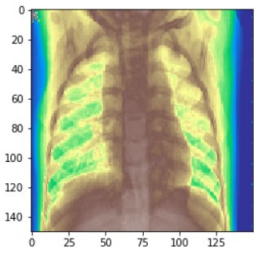


Figure 11 *Original colored image (with Matplotlib's terrain colormap)*

For image augmentation I used Keras's built in function (ImageDataGenerator) which augments the images in the memory and feed them to our deep learning model right away. This makes this task more efficient and faster than traditional image augmentation methods. On *Figure 11* you can see the original image (on which I already applied the 'terrain' colormap) and on *Figure 12* you can see some examples of the augmentation of this image.

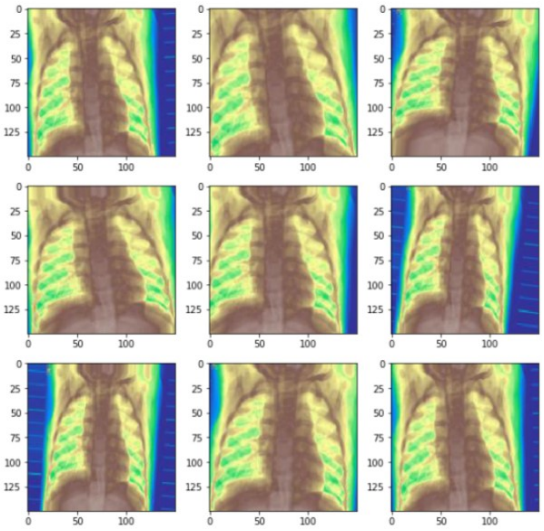


Figure 12 *Colored and augmented images (with Matplotlib's terrain colormap)*

Implementation

My solution for detecting Children Pneumonia from X-ray images was to use exclusively transfer learning with 3 different architectures: Xception, InceptionResNetV2 and InceptionV3. Repurposing these pre-trained models for my own needs was made of 2 main steps: defining a new classifier and fine-tuning the models.

Defining the classifier

The classifier part of each model contained a Dropout layer with 50% chance of getting dropped out, a Batch Normalization layer and a Sigmoid activation function. *Figure 13* depicts a sematic figure of the models' architecture. On the figure the input layer and the convolution blocks represent the predefined model architecture and the classifier and the output layer represent the personalized parts for this specific problem.

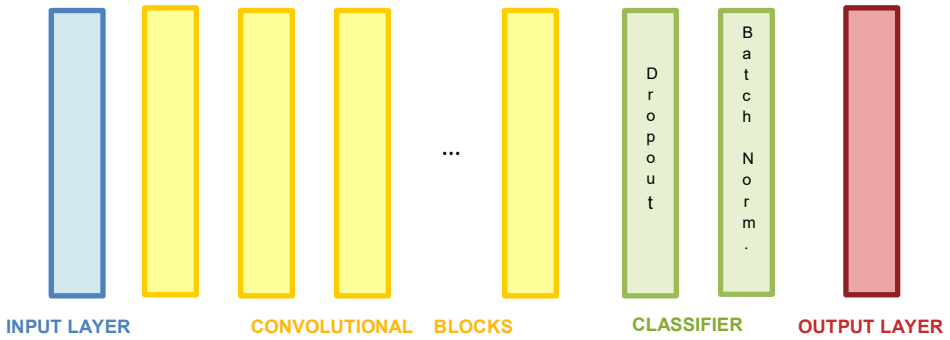
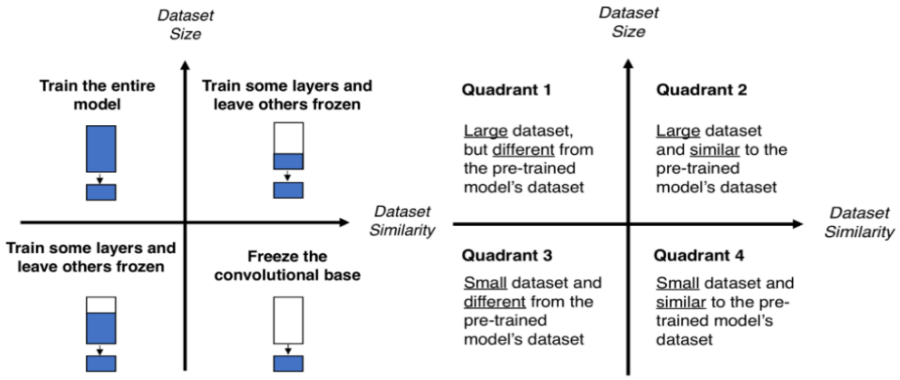


Figure 13 *The sematic structure of the models*

Fine-tuning the models

The fine-tuning of a model can be different according to the given problem. To help us decide which way to go, we can take a look at the Size-Similarity Matrix on *Figures 14 and 15.* This matrix classifies computer vision problems considering the size of the used dataset and its similarity to the benchmark dataset (ImageNet) on which the model was trained originally. (Marcelino, 2018)



Figures 14 and 15 *Size-Similarity matrix (left) and decision map for fine-tuning pre-trained models (right).* (Marcelino, 2018)

In my case I had a small dataset which is different from the benchmark. This means that the ImageNet dataset is not containing any class that would be similar to X-ray images. With the help of the Size-Similarity Matrix I decided to freeze some layers and to train the rest. Freezing a layer means that we keep (freeze) their weights, so we do not modify anything on them during the training process. This technique is quite effective as lower layers refer to general features (problem independent), while higher layers refer to specific features (problem dependent). Basically with choosing the number of layers to train we determine how much we want to adjust the weights of the network. Deciding on the number of layers to train is a hard job to do. My methodology was firstly, to train each model from the start and then I examined the architectures one by one and tried to find the adequate convolutional blocks where to end the freezing and start the training of the given network. *Figure 16* shows the layers from which the models were trained for each architecture.

Architecture	Trained from layer
Xception	0
Xception	75
Xception	95
Xception	115
InceptionResNetV2	0
InceptionResNetV2	59
InceptionResNetV2	287
InceptionResNetV2	630
InceptionV3	0
InceptionV3	40
InceptionV3	100
InceptionV3	196
InceptionV3	279

Figure 16 *Summary of models according to the layers they were trained from*

When training a neural network, we should also be careful with the learning rate we use. The learning rate is a hyper-parameter that controls how much we adjust the weights of our network. When we are using a pre-trained model based on a Convolutional Neural Network (CNN), it's smart to use a small learning rate because high learning rates increase the risk of losing previous knowledge. Assuming that the pre-trained model has been well trained, keeping a small learning rate will ensure that we don't distort the CNN weights too soon and too much. (Marcelino, 2018) During my training I used a learning rate of 0.001 which was decayed after each epoch when the validation loss did not decrease compared to the lowest validation loss so far.

Refinement

In general, I can confidently state that the all the architectures' performance (in the case each examined metrics) were getting better in the first 3 training processes (except for Inceptionv3 where it got better until the 4th training process) and after that the results started to go down (see more on *Figure 17*). I evaluated the performance of the models after each different training process with plotting the confusion matrix, printing the Precision, Recall, F1 Score and AUC.

The most important metric for me to decide on the best model was AUC as this was the only metric which can be properly compared with ChestXNet's result on Pneumonia.

Architecture	Trained from layer	Precision	Recall	F1 Score	AUC
Xception	0	0.92	0.97	0.93	0.9171
Xception	75	0.93	0.96	0.93	0.9201
Xception	95	0.91	0.97	0.92	0.9026
Xception	115	0.9	0.98	0.92	0.8979
InceptionResNetV2	0	0.92	0.98	0.93	0.9158
InceptionResNetV2	59	0.92	0.97	0.93	0.9167
InceptionResNetV2	287	0.88	0.99	0.91	0.8863
InceptionResNetV2	630	0.88	0.99	0.91	0.8863
InceptionV3	0	0.89	0.99	0.91	0.8923
InceptionV3	40	0.93	0.98	0.94	0.9303
InceptionV3	100	0.93	0.98	0.94	0.9312
InceptionV3	196	0.9	0.99	0.93	0.9073
InceptionV3	279	0.92	0.99	0.94	0.9231

Figure 17 *Result of each model on the test set per each examined metric. The red highlight indicates the best model.*

IV. Results

Model Evaluation and Validation

The final model's architecture is built up from the architecture of the InceptionV3 model (*Figure 18*) with a custom classifier. This classifier contains a Dropout layer with 50% chance of getting dropped out, a Batch Normalization layer and a Sigmoid activation function.

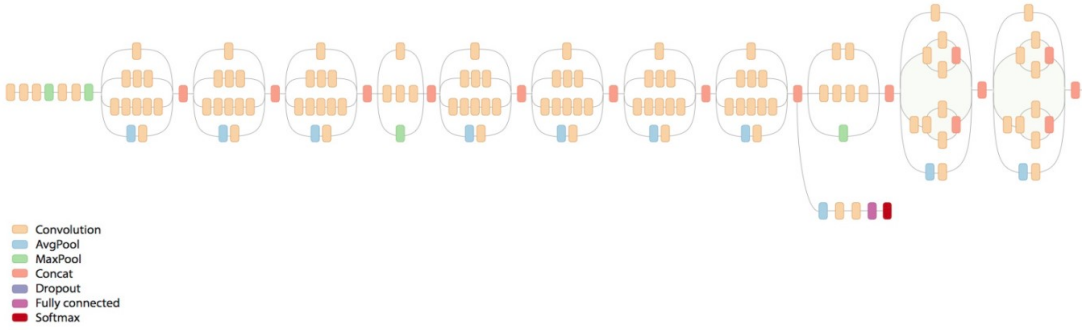


Figure 18 *InceptionV3's architecture without the last, classifier block*
Source: https://www.researchgate.net/figure/The-configuration-of-the-CNN-which-retrains-a-Google-Inception-V3-model_fig3_322216381

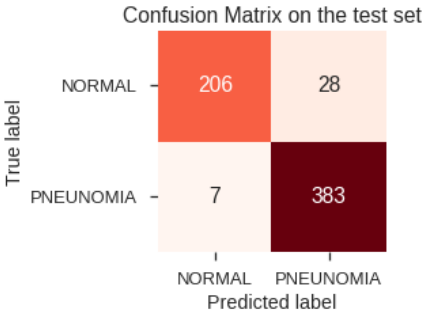


Figure 19 *The best model's performance on the test set plotted with confusion matrix*

We can clearly see on *Figure 17* that model based on InceptionV3 architecture was the best performing one when trained from the 100th layer. To make sure that this model generalizes well on unseen data, I evaluated its performance on the test set with plotting the confusion matrix alongside with printing the Precision, Recall, F1 Score and AUC values.

For more deeply understanding of the results, let's take a look at this model's confusion matrix calculated on the test set *(Figure 19).* We can clearly see that the model only misclassifies roughly the 6% of the images so this model generalizes well on each class. This means that with the usage of the model, the chance of giving wrong treatment to a healthy patient or not giving one to an ill patient is relatively small.

Justification

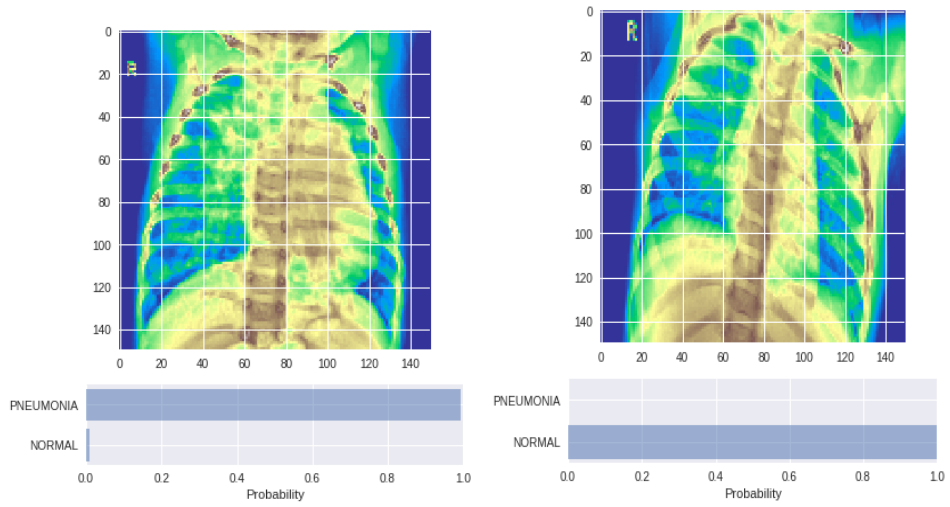
As we could see on *Figure 9*, the ChestXNet has a performance of 0.7680 AUC on Pneumonia prediction on the ChestX-ray14 database. My chosen best model has a 0.9312 AUC on the test dataset (*Figure 17)* provided by Kaggle. This means that my model's performance outperformed Stanford's ChestXNet model.

It is important to mention that this does not mean that my model is better than the ChestXNet. Both models were trained and tested on different datasets so their results cannot be comparable to each other. Their performance could only be comparable if they were tested on exactly the same images.

V. Conclusion

Free-Form Visualization

To emphasize the quality of my model I would like to present 2 random image examples (one for each class) from the test set I used. On *Figures 20 and 21*, under each image we can see a bar chart which tells us, how likely the given X-ray image is containing Pneumonia or not. We can see on both images, that the model predicts very well in each class.



Figures 20 and 21 *Test results on random images of Pneumonia (left) and Normal (right)*

Reflection

While working on my project I bumped into a few challenges. The first challenge was the coloring of the input images. It was tiresome to find a solution as I had this whole idea of coloring the black and white images somehow but could not find a proper solution for it. When I finally did, I just could not apply it only for one session without saving it to my computer. It took me several days to figure out the problem and find the best solution.

The next bigger challenge appeared when I was finally ready to train my models. First I tried to run it on my own machine, which took several hours. There was even a point when it took more than 1 day to run only one model. That was when I started looking for another solution where I can also use some GPU to make the training process faster. This is how I ended up training my models on Google's Colaboratory. At the end, the training of one model took maximum 20 minutes (with the evaluation part).

My final challenge appeared when I wanted my results to be reproducible so I tried to set a random seed. I was looking through a lot of solutions to set it properly, but none of them worked for me. It is hard to create reproducible results with GPU and my guess is that the problem is rooting from there.

Improvement

I think my Pneumonia diagnostic tool could be implemented into a software or into a webpage through an API where radiologists could easily access it. This tool could help them to diagnose Children Pneumonia better and faster.

I believe that my model's performance is not final yet. If we look back to the academic works of the past few years, we can see that each year there are always newer and even better solutions and methods for image recognition which constantly improves these models' accuracy. I strongly believe that some years from now there will be a model that can predict Pneumonia with 100% accuracy.

Bibliography

Amidi, S., & Amidi , A. (n.d.). *Blog of Shervine Amidi*. Retrieved January 05, 2019, from A detailed example of how to use data generators with Keras: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly

Bruce, P., & Bruce, A. (2017). *Practical Statistics for Data Scientists.* O'Reilly Media.

Marcelino, P. (2018, October 23). *Towards Data Science*. Retrieved from Transfer learning from pre-trained models: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

Mooney, P. (2018, March 24). *Chest X-Ray Images (Pneumonia)*. Retrieved October 10, 2018, from Kaggle: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia/data

Narkhede, S. (2018, May 9). *Towards Data Science*. Retrieved from Understanding Confusion Matrix: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., et al. (2017). *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays.* Stanford Machine Learning Group. Stanford University.

Shung, K. P. (2018, March 15). *Towards Data Science*. Retrieved from Accuracy, Precision, Recall or F1?: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

Wikipedia. (n.d.). *Clinical Decision Support System*. Retrieved 10 13, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Clinical_decision_support_system

World Health Organization. (2016, November 7). *Pneunomia*. Retrieved from https://www.who.int/news-room/fact-sheets/detail/pneumonia