Machine Learning Engineer Nanodegree

Capstone Project Chest X-ray Pneumonia Classifier

Ilona Brinkmeier, August 06th 2019

Content

Machine Learning Engineer Nanodegree....1
Capstone Project Chest X-ray Pneumonia Classifier....1
I. Definition....2
Project Overview....2
Problem Statement....3
Metrics....4
II. Analysis....6
Data Exploration....6
Exploratory Visualisation....8
Algorithms and Techniques....9
Benchmark....12
III. Methodology....12
Data Preprocessing....12
Implementation....13
Refinement....18
IV. Results....20
Model Evaluation and Validation....20
Justification....22
V. Conclusion....24
Free-Form Visualisation....24
Reflection....25
Improvement....26
VI. Addendum....27
Abbreviations....27
Other Model Architectures....28
Detailed Metric Results of the Selected Models Set....35

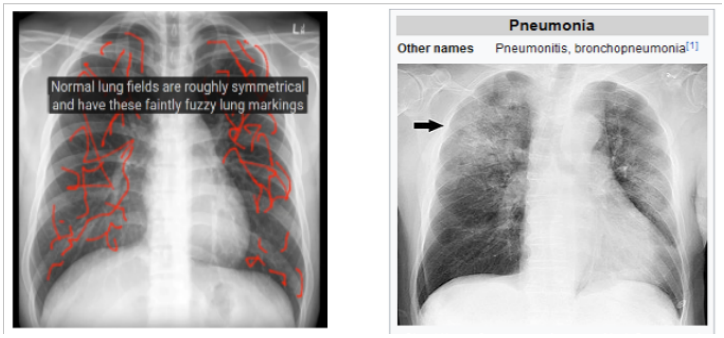
I. Definition

Project Overview

In general, to solve a business problem the overall solution process starts with business understanding, followed by understanding the existing data[1].

This capstone project is chosen from the medical healthcare area, because artificial intelligence algorithms have the potential to change the medical workflow being common in the future[2].

The implemented deep learning concept of Convolutional Neural Networks (CNN) shall deliver predictions of the binary classification problem if chest X-ray images are being normal or pneumonia ones.



First image source, second image source

Pneumonia is an infection of the lungs caused by microbes mainly. As stated by the World Health Organisation (WHO)[3], „pneumonia is the single largest infectious cause of death in children worldwide, accounting for 15% of all deaths of children under five years old.[...] Pneumonia caused by bacteria can be treated with antibiotics, but only one third of children with pneumonia receive the antibiotics they need."

Regarding this status about children mortality having a pneumonia, the used Kaggle dataset and an improved, final coding for having a fast, automatically created diagnosis would save lifes of the children triggering proper, faster treatments. For a non-radiologist it is hard to see the classification difference and more specific for the pneumonia cases, the root cause difference.

For a child having a pneumonia, this may lead to death, especially if the immune system is already compromised, e.g. bei HIV, tubercolosis or subnutrition. According the mentioned WHO fact-sheet „Pneumonia affects children and families everywhere, but is most prevalent in South Asia and sub-Saharan Africa." There the healthcare coverage of the whole area is not the same compared to western countries like in Europe or North-America. Having an automatic estimation would help to trigger necessary medical or other interventions, even when no radiologist is available.

With such kind of algorithm the general medical workflow would be improved as well: In a general hospital, radiologists are doing this classification manually on their viewing stations connected with

[1] Image source: SAP course 'Getting started with Data-Science'
[2] https://www.healthcatalyst.com/prescriptive-analytics-improving-health-care
[3] WHO & Pneumonia, https://www.who.int/news-room/fact-sheets/detail/pneumonia

the picture archiving and communication system (PACS modality)[4]. A medical PACS system is part of the hospital information system where all images are stored in a so called DICOM format (Digital Imaging and Communications in Medicine)[5]. The international *Digital Imaging and Communications in Medicine* standard, DICOM standard for short, delivers the processes and interfaces to transmit, store, retrieve, print, process, and display medical imaging information between relevant modality components of an hospital information system.

The classification deep learning method can ease the medical workflow and increase the diagnosis correctness rate, because an estimation can be given automatically and efficiently for thousands of images in a short time, even difficult ones not easy to diagnost manually. As a whole, this would not be possible for a radiologist: during a long time viewing process being more fatigue and unfocussed is a normal human reaction and therefore correctness may decrease.

Additionally, today there is already a shortage of needed radiologists. Each existing one has much more work to do than in former times. As mentioned in the 'AI in Healthcare' journal[6], for radiologists artifical intelligence methods like deep learning are going to „expedite and improve their ability to interpret images“.

Problem Statement

As a **summary**, the goal of the project is not to have a realy usable application, it is to find a proper deep learning CNN model architecture for the Chest X-ray images binary pneumonia classification problem, having at least nearly 86% test accuracy. This value has been mentioned by a TÜV (regulatory authority) speaker as maximum prediction value for this pneumonia classification task on a medical AI workshop in Germany happend on July 5th 2019 in the German city of Konstanz (organiser: Johner Institute[7]).

Input of few different CNN network model architectures are converted .jpeg compressed images based on their original chest X-ray .dcm DICOM images. The algorithm will identify an estimate of the image status showing a pneumonia or not. Build and find a neural network for a specific task is not trivial. To predict which architecture structures are the best ones for this classification task is not easy. Therefore the different models are compared and evaluated with each other by some metrics, relevant for the binary classification or the dataset distribution.

The used CNN model hyperparameters are considered to be a 'starting point'. It is expected by doing a detailed hyperparameter tuning to get better, stable prediction results (from Scikit-Learn with GridSearchCV or RandomizedCV and the implementation of adapted-learning-rate values and the early-stopping concept). This has not been possible now, because of project deadline and its needed high amount of time. It is computational expensive having such a lot of parameters for the neural network architectures.

An already existing solution is described in the Cell journal paper 'Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning'[8]. There as well, deep learning CNN models have been implemented, by using the official ImageNet database from www.image-net.org as input for transfer learning techniques and as a final result, delivering specific classification estimations of having a bacteria or virus root cause of the pneumonia.

---

4 Acronym with several meanings, here in the medical domain: https://en.wikipedia.org/wiki/Picture_archiving_and_communication_system

5 DICOM - https://www.dicomstandard.org/

6 https://trimed-cdn.s3.amazonaws.com/digitalissues/aih/2019/2019_02/index.html article 'Embrasing AI: Why Now Is The Time For Medical Imaging'

7 https://johner-institute.com/ ; German link of the workshop: https://www.johner-institut.de/institutstag/

8 https://www.cell.com/action/showPdf?pii=S0092-8674%2818%2930154-5

Nevertheless, have in mind, that this Python classifier coding cannot be used in real life, improvements would be necessary as well as mandatory regulatory aspects, which have not been solved yet.

Metrics

This project deals with a classification problem therefore for the CNN models as metric 'Accuracy', 'Confusion matrix' and others can be used, considering the prediction result of the task and the distribution characteristic of the Kaggle dataset.

- **Training- and Test-Accuracy**

  Accuracy measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions

  (TP + TN) / (TP + TN + FP + FN)

  where: TP = TruePositive; FP = FalsePositive; TN = TrueNegative; FN = FalseNegative

- **Learning rate** of the training and validation dataset samples

  In general, the training loss is the average of the losses over each batch of training data. There are 32 samples in each batch, which are processed independently. The model 'learns' over time, so regarding an epoch, at the beginning the learning loss value from the first batch is higher compared to the loss value of the final batch at the end. During validation, only the loss value of the epochs end is created, means a lower value shall exist.

  The learning curves shall be smooth and decreasing for both datasets nearly in the same way.

- **Precision**

  Precision quantifies the binary precision. It is a ratio of true positives (images classified as pneumonia ones, and which are actually pneumonia) to all positives (all images classified as pneumonia ones, irrespective of whether that was the correct classification), in other words it is the ratio of TP / (TP + FP).

- **Recall**

  Recall, also called sensitivity, tells us what proportion of images that actually were pneumonia ones were classified by us as pneumonis ones. It is a ratio of true positives to all the images that were actually pneumonia ones, in other words it is the ratio of

  TruePositives / (TruePositives + FalseNegatives)

- **F-beta score (F-β)**

  A model's ability to precisely predict those that have a pneumonia is more important than the model's ability to recall those individuals. **F-beta score** is used as a metric that considers both precision and recall. According scikit-learn, the F-β score is the weighted harmonic mean of precision and recall, reaching its optimal value at 1 and its worst one at 0.

  F-β = (1 + β2) (precision·recall / ((β2·precision) + recall))

  F-beta score with beta 0.5: when β=0.5, more emphasis is placed on precision.

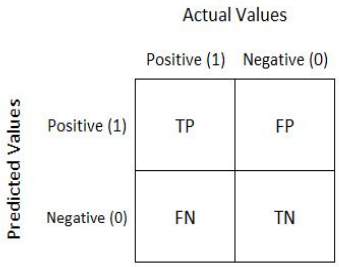
- **Cohens Kappa**

  As realised, the dataset is imbalanced, because mostly children images exist and more images with bacteria root-cause. Does this have an effect on the prediction? Are the training, validation and testing sets are constructed properly? In the medical domain, the natural

frequence of the class or events is normally imbalanced. Cohens Kappa is a statistical measure of the inter-rater agreement. It can be used to get information about the imbalance of the dataset. In general, Cohen's Kappa takes the class distributions of the (training) data into account. Kappa values are in the range of [-1, 1] and 1 shows a perfect match between the predicted class and the observed one. Values between 0.3 and 0.5 are interpreted to be reasonable[9].

- **Confusion Matrix** of the test data

  It is a common method to get information about the classification performance. It looks like



  where positive maps to the pneumonia cases, negative to the normal ones.

  There are two types of errors that can be identified here:

  Type 1 Error: The model predicted the instance to be a Positive class, but it is incorrect. This is FalsePositive (FP).
  Type 2 Error: The model predicted the instance to be a Negative class, but it is incorrect. This is FalseNegative (FN).

  Having a medical diagnostic prediction task, this type 2 error (FalseNegatives (FN)) is more important. Inaccurately classified patients have a high increase of the worst case risk to die, because of its severe consequence for the undetected ill patient, regarded being healthy and not getting the needed therapy and being unmedicated.

- **ROC AUC** diagram including the random line caused by chance

  The ROC – Receiver Operating Characteristic - curve is a curve between true positive and false positive rate for various threshold values. It is telling us about the 2 types of errors for all thresholds. The more the area under the curve (AUC), the better is the performance of our model to distinguish the classes. Means, it is a prediction probability curve for the different classes. It informs us how good or bad our overall model performance is, mapping the *conditional* measures sensitivity and specificity into one value. The ROC AUC diagram is used to compare the different classifier models comparing it with a random option existing by chance and visualised by a diagonal, dotted line (50% cuttoff).

  A typical ROC curve has False Positive Rate (FPR) on the X-axis and True Positive Rate (TPR) on the Y-axis. And the ideal value for AUC is 1, means having a perfect classification. In general, such best case scenario will not be reached, means we don't have a clear distinction between the two classes NORMAL and PNEUMONIA. Overlapping of the class predictions exists and that introduces type 1 and type 2 errors to the model prediction.

Some of these metrics are more suitable for our medical prediction task. Only accuracy will not fit to the complexity of real life situation. Nevertheless, trade-offs are necessary to get the right model for the classification performance. We want to know the learning rate, because it shows the model training costs. For comparing the models qualities the confusion matrix (mainly false negatives) and

[9] According 'Applied Predictive Modeling' from Max Kuhn, Kjell Johnson; chapter 11.2 Evaluating Predicted Classes

the ROC AUC (for class probabilities) are important. Having a look to the dataset, Cohen's Kappa shall have an acceptable value and not too small.

II. Analysis

Data Exploration

The used Kaggle dataset[10] delivers already labelled images as training, testing and validation samples. Each of this directories includes the two sub-directories 'NORMAL' and 'PNEUMONIA':

- PNEUMONIA: Samples of pneumonia cases including only virus and bacteria root causes
- NORMAL: Samples that describe the normal (no pneumonia) cases

The dataset cannot properly be structured in age ranges for babies, toddlers, younger children, teenagers or adults. The jpg images are not from equal size, sizes are very different and they have:

- resolution: 96 dpi (Dots per Inch), for checked baby, child or adult images
- bit depth for all checked images: 8 (means 2^3)

The images are looking as greyscale ones, but the tensors creation of the TensorFlow software sets the channels parameter to 3 and this stands for RGB (Red Green Blue) colour image. It is possible that RGB is converted to a greyscale sequence or vice versa. X-ray images, created according DICOM standard, are greyscale images with higher bit depth having a greyscale sequence from white to black (e.g. according attributes for the 'Basic Grayscale Image Box SOP Class')[11].

Dataset Distribution

For the project, the dataset images are modified on 2 additional datasets used for investigation, because the original dataset from Kaggle doesn't fit to the rules-of-thumb according distribution ratios for training-validation-testing sets. So, we apply 3 dataset distributions for investigation:

- the original dataset distribution with the following ratio for train, validation and test directories and its percentages

```
--- Train files ---
1341 normal chest images.
3875 pneumonia chest images.

--- Validation files ---
8 normal chest images.
8 pneumonia chest images.

--- Test files ---
234 normal chest images.
390 pneumonia chest images.
```

```
There are 2 total chest categories
There are 5856 total chest images.

There are 5216 training chest images: 89.071%
There are 16 validation chest images: 0.273%
There are 624 test chest images: 10.656%
```

- V2 distribution of nearly 70-10-20 percent

```
There are 4224 training chest images: 72.119%
There are 476 validation chest images: 8.127%
There are 1157 test chest images: 19.754%
```

- V3 distribution of nearly 60-20-20 percent

```
There are 3520 training chest images: 60.079%
There are 1168 validation chest images: 19.935%
There are 1171 test chest images: 19.986%
```

[10] https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia/version/2

[11] Own medical and DICOM knowledge; official page e.g. https://www.dicomlibrary.com/dicom/dicom-tags/ https://en.wikipedia.org/wiki/Grayscale#Grayscale_as_single_channels_of_multichannel_color_images

As mentioned, such Kaggle images are based on associated chest X-ray DICOM images which are not available. The jpeg-transformation is the reason why private individual data information, as existing in the DICOM images, don't exist anymore. After viewing such images manually, it has been identified, that posterior-anterior or anterior-posterior X-ray image orientation is available, but no lateral ones. Mostly children and pneumonia images are selected.

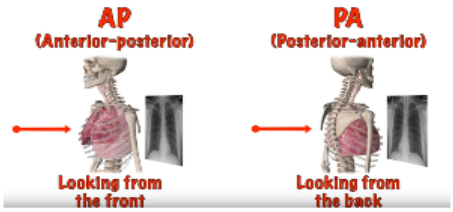


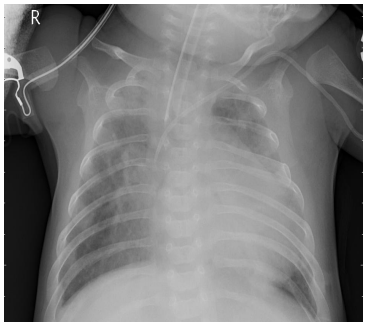
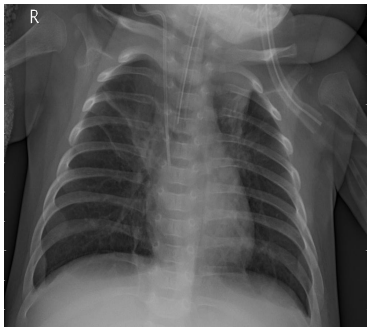
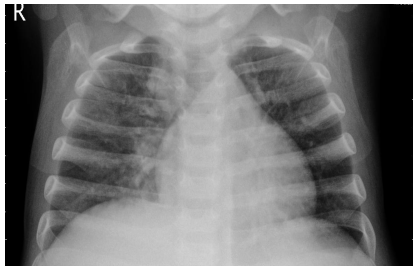
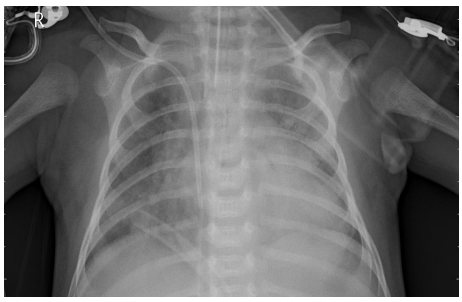
Image source

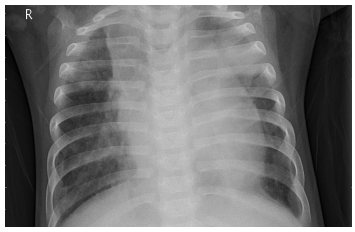
The age and orientation status could only be analysed more properly by reconverting the images to the .dcm DICOM format having the associated DICOM tags available. Doing this, regulatory data protection aspects have to be taken into account (e.g. Health Insurance Portability and Accountability Act, HIPAA[12]), therefore this has not been done. It would be a HIPAA compliance breach of this project.

In other words, the dataset is imbalanced, but with respect to the WHO information and using the dataset to improve the situation of the children first, this should be no issue for the investigation. As mentioned, it is a normal situation for the medical domain and not severe in this case.

According the Kaggle context „the normal chest X-ray depicts clear lungs without any areas of abnormal opacification in the image. Bacterial pneumonia typically exhibits a focal lobar consolidation, [...], whereas viral pneumonia manifests with a more diffuse ‘‘interstitial’’ pattern in both lungs.“ Are you able to see this differences in the following example images of the dataset?

Here are some **pneumonia images**: 3 virus images, followed by 3 bacteria images



And now having a look to 3 normal patient images (**non-pneumonia ones**):

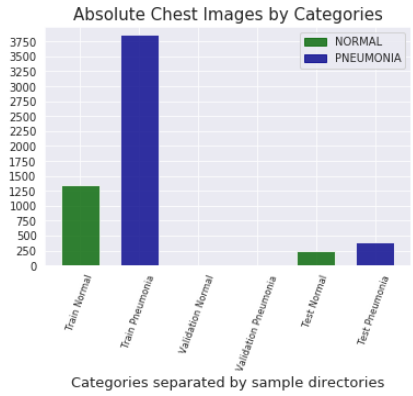
[12] https://hipaa.com/

It is hard to see the difference between non-pneumonia and pneumonia images and furthermore, differentiate the root cause with the existing X-ray images. Additionally, the wide range of image quality aspects (e.g. brightness and contrast) increases this situation. Therefore, having an application to help the medical experts to do the diagnosis would be a great benefit.

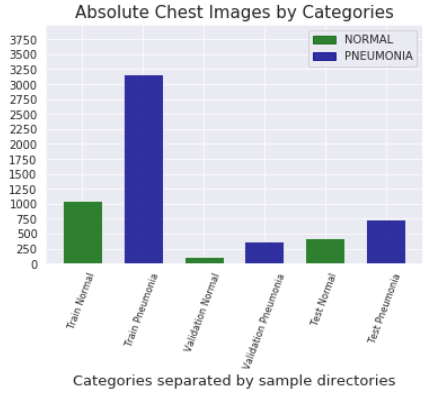
Exploratory Visualisation

The 3 different dataset distributions are visualised now to extract their characteristics. We start with a general overview:

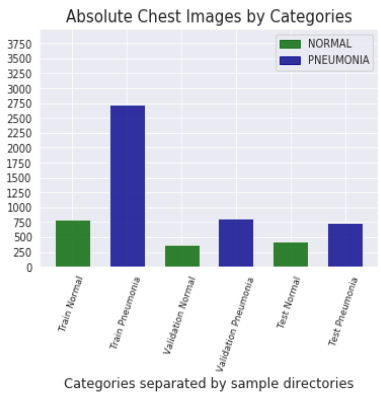
Original Dataset



V2 70-10-20



V3 60-20-20

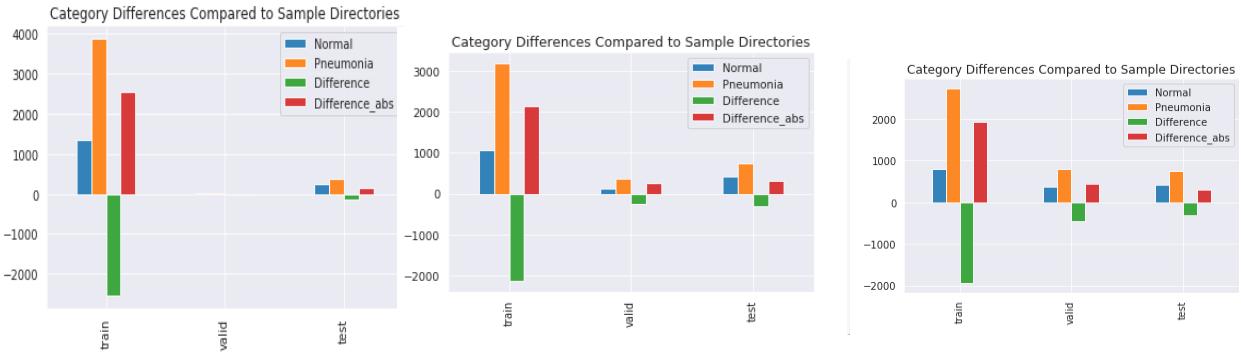


As we can see for each distribution, there are much more pneumonia images compared to normal cases. But it is important for the model to learn what is a pneumonia image and what isn't, therefore this is regarded to be fine. For the original dataset, especially for the training set, the ratio normal:pneumonia is 1:2.89. Furthermore, its amount of validation samples is much too small, therefore no columns are visible.

The distribution of the original dataset does not fit dataset rules-of-thumb for training-validation-testing splitting. Beside the general rule of having training/testing ratios of 80/20 or 70/30, the amount of validation and testing samples shall be the same. Therefore the other 2 distributions have been build: as a trade-off between the original setting and this rules version 'V2 70-20-20' and foccusing on the second rule the version 'V3 60-20-20'. Such percentages could be nearly reached, but is not exactly, because of the batch size configuration for the neural networks.

Now, the relationship of the difference amount value between the normal and pneumonia samples are visualised.

Original Dataset V2 70-10-20 V3 60-20-20



Changing the distributions has a most prominent effect on the validation set, as visible with the (absolute) difference value of the above diagrams. The training sets doesn't seem to change much, but having a closer look to the scaling and the difference columns, it is visible that the imbalance between normal and pneumonia images is going down with distributions V2 and V3.

Finally, as mentioned there is another information of the chest images regarding the pneumonia root cause. There are several reasons of getting a pneumonia, here only 2 are available. For the original dataset it is:

```
--- Train pneumonia files ---
1345 virus chest images.
2530 bacteria chest images.

--- Validation pneumonia files ---
0 virus chest images.
8 bacteria chest images.

--- Test pneumonia files ---
148 virus chest images.
242 bacteria chest images.
```

For the investigated models such information isn't used. It is only an additional information of the datasets. It would be a future to do, to use this additional classification topic for a real application.

Algorithms and Techniques

The answer to the question of having a pneumonia chest X-ray image or not, is technically a binary classification issue. Because we are dealing with **images, convolutional neural networks** are a state-of-the-art solution concept for such kind of task[13]. Having properly labelled datasets for training, testing and validation of the neural networks, such CNNs map image data – spatial relationships are taken into account – to our desired prediction estimation as output value.

The general architecture of CNNs looks like[14]

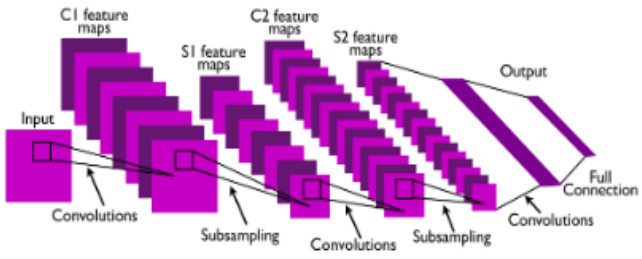


Fig. 1. A typical ConvNet architecture with two feature stages

[13] Paper 'Going deeper with convolutions' by Christian Szegedy, Google Inc., et all. https://arxiv.org/pdf/1409.4842.pdf

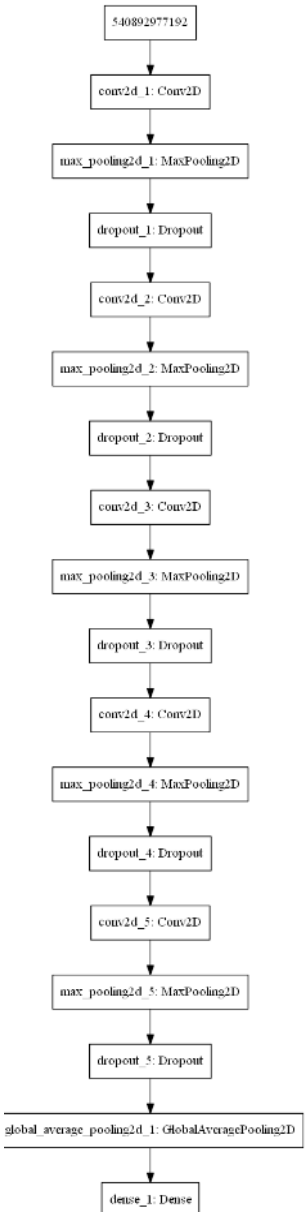
[14] Image source: http://yann.lecun.com/exdb/publis/pdf/lecun-iscas-10.pdf

In general, the network consists of an **input layer** for the image, blocks of **hidden layers** inluding the feature map arrays and the **output layer** for the estimated class prediction value. For the investigated CNN models the hidden layer architecture makes the difference where the main computation is done including calculated node weigths. Such weights are influencers for the nodes of the next layer and its weighted sum is input part of a so called 'activation' function, predefined for the network calculations. This function defines if and how a node is activated. The **weights matrix** of a neural network is usually labelled as θ or W. So, to get the 'activation' nodes of the hidden layer architecture, e.g. for the first hidden layer block: the input vector X and weights matrix W for the first layer have to be multiplied, followed by the activation function usage. Doing this for each layer block, a generalised node value function is computed.

All this happened in the background with the used Keras[15] and TensorFlow libaries by setting the specific network configurations. Beside the activation function, other predefined parameters for the CNN model exists. Some of them are investigated with the goal to get an improved classification performance.

A basic architecture is the starting point to compare its results with the other models results. Unfortunately, it has not fit to the expectations regarding its training- and learning curves and also its metric results (see addendum chapter about the detailed metric results of selected models set).

```
--- Build model summary of Basic_CNN_Model: ---
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 128)       32896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 256)       131328
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 7, 7, 256)         0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 514
=================================================================
Total params: 175,282
Trainable params: 175,282
Non-trainable params: 0
_________________________________________________________________
```



[15] The Keras API can be find with https://keras.io/

The hidden layer is a combination of several *Convolutional*, *MaxPooling* and *Dropout* subblocks, then finished by a *GlobalAveragePooling* flatten- and *Dense* output-layer representing the none and the pneumonia classes. In other words, e.g. Dropout layers are already included to decrease the risk of overfitting.

For the internal subblocks the 'relu' **activation function** and for the output part, having 2 output classes and their prediction values, the 'softmax' activation function is selected (coding: Dense(2, activation='softmax')).

Another important configuration for the network is the **cost or loss function**. This function represent the sum of the error, means the difference between the predicted class value estimation and the existed labelled value for the image. This error shall be minimised by using the optimal set of weights *W*. As common, the Keras coding implemented the **backpropagation** concept to calculate the partial derivatives for *W* to minimise the cost function.

As cost or loss function Keras offers few, it is needed to compile the model. For the project, having a binary classification task, we are using 'binary_crossentropy'.

Next parameter we have to look for is the **optimiser**, which helps to find the general, global optimum of the architecture or a good accepted local one with low costs. This is necessary, because with our network surface, we have several optima and don't want to get stuck in a bad local one with high costs.

Here as well Keras offers few possibilities, and together with the loss function, it is a mandatory parameter for compilation too. With our basic model we are starting with 'rmsprop' as optimiser, using some improvement techniques afterwards.

Regarding this optimum topic, we have to look for an appropriate **learning rate** (lr) as well. If it is too small, to find the optimum takes too long; if the learning rate is too big, no converge will happen. We are starting with the Keras default of the rmsprop optimiser:

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

Model training happens with use of **checkpointing** to save the model that attains the best validation loss. The following hyperparameters are needed too for network training:

- `epochs` is set to a number of iterations, it shall include the point where the network stops learning or start overfitting
- `batch_size` is set to a common minimal size of memory: 32 (if your machine is better use 64, 128, 256, ...) For the original dataset: training on 5216 samples, therefore batch_size=32 (5216/32 = 163)

Afterwards some **more complex CNN** model architectures are investigated. Their architecture diagrams are put to the addendum chapter of this report:

- A *self-created model* having more convolutional hidden layers compared to the basic model and using Adam as optimiser at once.
- Using the transfer learning method, creating a pre-trained *ResNet50* CNN model by Keras. ResNet is a short name for residual network. It includes a lot of subblocks with BatchNormalization layers.
- Keras *InceptionV3* CNN model is one of the models to classify images. The concept of the inception module focuses on the kernel size of convolutional layers as e.g. described in the already mentioned paper 'Going deeper with convolutions'.

For both, last transfer learning models (ResNet50, InceptionV3), as 'top model construction' a

flatten *GlobalAveragePooling2D* layer, followed by a *BatchNormalization* instance and the fully connected Dense layer, which handles the 2 different chest categories, is implemented.

Benchmark

Regarding the 3 used distributions and their metric results for the basic model architecture, they are selected to benchmark the other investigated model architectures. Additionally, as estimation treshold a TÜV representative mentioned test accuracy of 86% is selected. As expected, the basic model with default parameters has much lower accuracy values for each distribution. Furthermore, we want to reach at least nearly this 86% value by improvements. To decide which model is the best, other metrics, especially the confusion matrix, Cohen's Kappa and the ROC AUC diagram are important and described in the 'IV Results' chapter of the report. Investigation details are documented in the addendum part 'Detailed metric results of the selected models set'.

Regarding the accuracy only for the benchmark model, the following has been investigated as basic values:

- **Original dataset**

  Accuracy for training and testing of the basic model with rmsprop optimiser:
  `Train: 74.3%, Test: 62.5%`

- **V2 70-10-20 dataset**

  Accuracy for training and testing of the basic model with rmsprop optimiser:
  `Train: 75.2%, Test: 63.4%`

- **V3 60-20-20 dataset**

  Accuracy for training and testing of the basic model with rmsprop optimiser:
  `Train: 77.5%, Test: 63.3%`

III. Methodology

Data Preprocessing

Data Distribution

Based on the Data Exploration chapter, one part has already been described – the creation of modified dataset distributions, because the original dataset does not follow the rules-of-thumb. So, images are sorted again by random to training-validation-testing sets via new ratios. With the Python notebook coding, this specific split information is stored as .csv files for the training-validation-testing sets of each distribution in the newly created 'requirements' project directory.

Image Convertion

Keras is using TensorFlow as backend, means as tensor manipulation. According the TensorFlow guide[16], a "tensor is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base datatypes."

Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input for training and testing with shape (nb_samples, rows, columns, channels), where nb_samples corresponds to the total number of images (or samples), and rows, columns, and channels correspond to the number of

[16] https://www.tensorflow.org/guide/tensors

rows, columns, and channels for each image, respectively. The channels value is 1, if the image is a grey-scale image and 3 otherwise (RGB image as in this project).

First, the image load happens, then it is resized to a square image that is 224×224 pixels. Next, the image is converted to an array, which is then resized to a 4D tensor. Likewise, since we are processing a single image, the returned tensor will always have shape (1, 224, 224, 3). Afterwards, we rescale the images by dividing every pixel in every image by 255.

Implementation

For each data distribution a separate Python 3.6 Jupter notebook implementation is created:

- chest-class_app_originalData.ipynb
- chest-class_app_V2_70-10-20.ipynb
- chest-class_app_V3_60-20-20.ipynb

All models are implemented in their own class file and not directly in the notebook. All additional evaluation functions to calculate the metrics or plotting diagrams are separated in its own class as well.

The main part of each notebook implementation, after pre-processing, is the training section of the few different CNN models, followed by the testing phase and its metrics evaluation.

As **model types** we compare:

- **Basic CNN Model** from scratch, the one that is already introduced in the 'Algorithms and Techniques' chapter. Start, creating this basic model with 5 blocks of *Convolutional*, *MaxPooling* and *Dropout* sublayers as hidden layer components. Dropout layers have already been added to reduce overfitting. As flatten layer *GlobalAveragePooling2D* is used.

  Parameters are used with Keras default configuration and according own environment:

  - `epochs = 20, batch_size = 32  (not higher because of weak environment)`
  - `optimiser: RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)`
  - `output Dense layer: 2 resulting categories NORMAL and PNEUMONIA using 'softmax' as activation function`
  - `compilation code is then (loss and metrics is the same for all models):`
    ```
    base_model_class.compile_model(model=base_model, optimizer='rmsprop',
                                   loss='binary_crossentropy',
                                   metrics=['accuracy'])
    ```

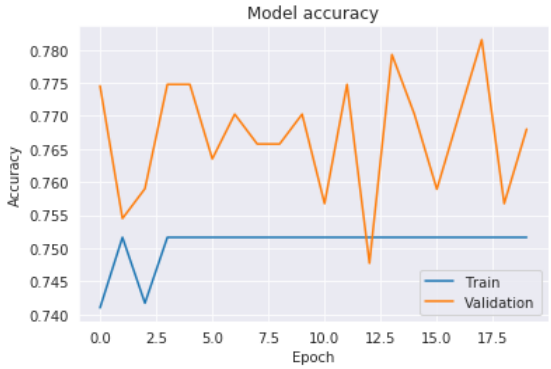
  Improvements are done on this model using the augmentation concept, Adam as more appropriate optimiser parameter as state-of-the-art today and then both improvments at once. With the augmentation concept, creating new image samples by changing some image properties 'that fit to the medical X-ray images', we get a more robust training. Keras has introduced an *ImageDataGenerator* object to do this:

  ```
  train_data_generator = ImageDataGenerator(
         rotation_range=20,
         width_shift_range=0.2,
         rescale=1.,
         height_shift_range=0.2,
         shear_range=0.2,
         zoom_range=0.05,
         horizontal_flip=True,
         vertical_flip=False,
         fill_mode='nearest')
  ```

  The training results of the first model from scratch with rmsprop optimiser showed already

that learning stopped already after first epoch with e.g. for the original dataset val_loss = 5.94746, which is a much too high value to reach the first benchmark topic of 86% test accuracy. The change of the optimser to Adam showed a big improvement effect, more than augmentation alone. E.g. for the modified dataset V2 70-10-20 the accuracy results are:

```
Base model with rmsprop optimiser and augmentation:
```



By using the augmentation concept,there is already a good improvement visible, but it is still a biased dataset distribution: The validation curve is much higher compared to the training one. So, this concept is not enough to get rid of the underfitting issue. Together with Adam optimiser we get

```
Basic, augmented model with Adam optimiser:
```

Using both concepts - Adam optimiser and augmentation - the situation changes after epoch 3. Then the opposite, means overfitting is available.

As a final insight for this basic model: in general, best results are reached with Adam and augmentation for all 3 datset distributions. Nevertheless, as shown in the 'Justification' chapter, this model type is not suitable for the classification prediction task.

- **Improved CNN Model:** Thus, having a more complex hidden layer architecture including more convolutional layers, would this be an improvement getting better results? The architecture includes 6 hidden-layer blocks like:

```
self.model.add(Conv2D(filters=64, kernel_size=2, padding='same',
                      activation='relu'))
self.model.add(Conv2D(filters=64, kernel_size=2, activation='relu'))
self.model.add(MaxPooling2D(pool_size=2, strides= 2, padding='same'))
self.model.add(Dropout(0.3))
```

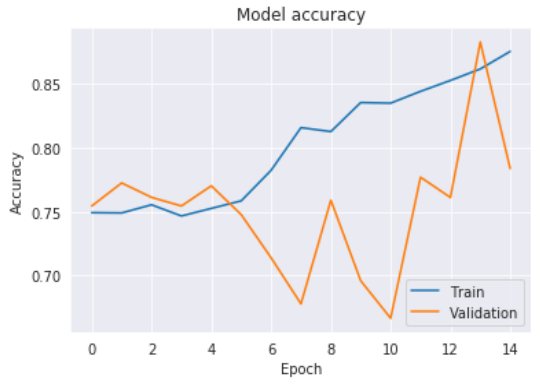
So, there are some more Conv2D layers and one additional complete subblock compared to the basic model and Adam is used at once. Furthermore, Keras default for the optimiser

```
Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0,
     amsgrad=False)
```

has been changed according weight decay to avoid overfitting much more than only using Dropout nodes by usage of weight decay 1e-6 for each learning update.

But the expectation was wrong for all 3 dataset distributions, even by using the improvement augmentation concept too. E.g. for the modified dataset V2 70-10-20 the confusion matrix of the test data showed an unacceptable high value of false negatives (77) and its test accuracy is only 82.8%, showing overfitting in the epochs sequences:

```
Improved, augmented model with Adam optimiser:
```

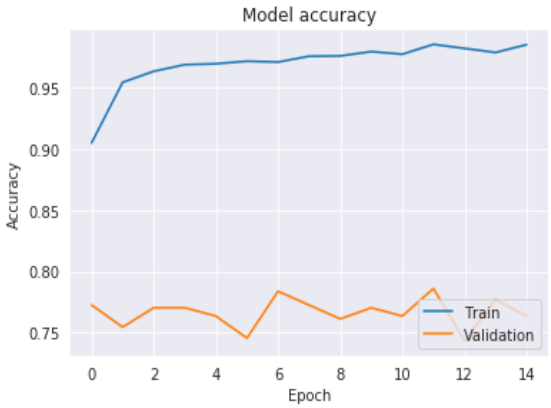


What would be the classification prediction results by using the transfer-learning concept?

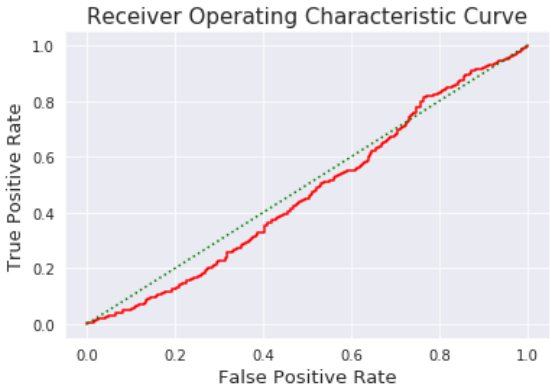
- With the **RestNet50** transfer model using **bottleneck features** (together with Adam optimiser, with weight decay 1e-6, and a smaller learning rate lr=1e-4 to avoid to get stuck in a local minimum and to reduce overfitting again) a fast high training accuracy appeared and the validation accuracy follows in the same way. This model type has been the best for all 3 dataset distributions as described in the 'Result' chapter. The complex architecture diagrams are visible in the addendum of the report.
- Not knowing, that the former one is the best, the question pops up, if a this model could be improved to make it a 'fine-tuned' one: We unfreeze the last res5c layer block. There, the layer weights are updated in each epoch as we pass the data batch. The training is implemented to use the augmentation concept with generators as before.

  But e.g. for the modified dataset V2-70-10-20 its test accuracy has only been 63.4%, showing overfitting the whole epochs sequences.

```
Fine-tuned ResNet50 transfer model with Adam optimiser and augmentation:
```



And its ROC-AUC curve showed, that this model has nearly the same class probabilities as doing prediction by chance:

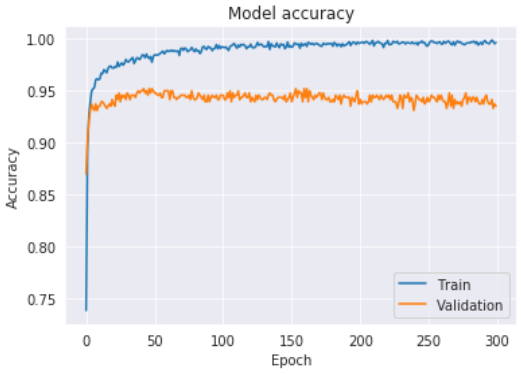


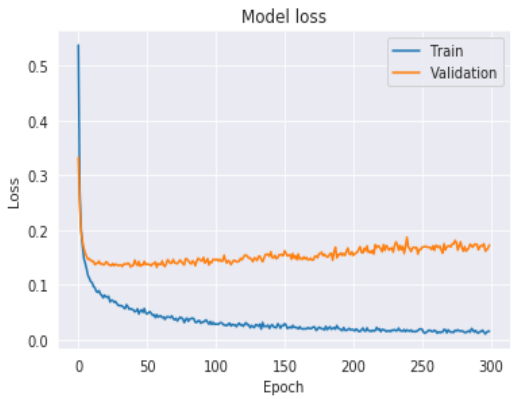
So, as an unexpected project insight, there is no general rule always having an accuracy improvement by using a fine-tuned transfer model.

- Another transfer-learning model is the **InceptionV3**. The concept of the inception module focuses on the kernel size of convolutional layers as e.g. described in the mentioned paper 'Going deeper with convolutions'. Is the InceptionV3 model used with bottleneck features better compared to the ResNet50 model used with bottleneck features? Again, we create a pre-trained InceptionV3 CNN model and freeze the convolution blocks, so that we can use it as an image feature extractor to get the bottleneck features. The models complex architecture is shown in the addendum as well.

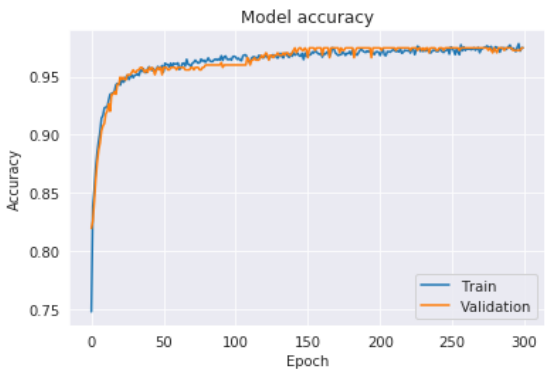
  Again, for the V2 70-10-20 dataset it has a good test accuracy of 86.34%, but 15 false negatives which is high compared to the 5 of the ResNet50 model. And for both models, comparing the accuracy and learning diagrams the ResNet50 wins.

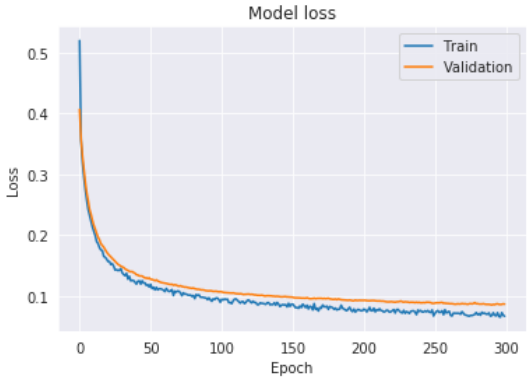
InceptionV3





ResNet50





Even using the Keras and TensorFlow libraries, some unexpected difficulties appeared:

Training time range

Even investigating only some models and improvement techniques the training time is very computational expensive. So, by using only older CPU hardware (IntelCore i5 processor) not made for data science tasks, it could take a week for one notebook file. Using an Amazon AWS EC2 service and having a stable internet connection, training for one distribution took nearly 2 days.

From the final model types, the fine-tuned RestNet50 with augmentation took much, much more training time as all other ones. Most part of the image has already been pre-trained, only the last 9 nodes are activated again to be used for fine-tuning, so, it has been expected to be ready fast. This together with augmentation took several hours more to train. The original ResNet50 used with bottleneck features was ready in several minutes and this by having the highest used epochs configuration. There most time has been spent to create the bottleneck features.

Therefore, for the fine-tuned ResNet50 models of the modified datasets the epochs size has been changed from 20 to 15, after having such experience with the very slow training of the original dataset.

Finally, because of that already existing huge amount of time to train the models, it has been decided to skip the investigation of an own created model using BatchNormalization nodes heavily (it would take hours to train too), supported by the knowledge that e.g. the ResNet50 model includes such a lot of BatchNormalization nodes in all hidden layer subblocks, that this official model could not be beaten by a good performance of such a very simple own one.

Training learning process

Not all models show a real training and learning process, visible in the associated curves of the Evaluation section and in this cases, the metrics are not properly calculated. Especially the confusion matrix calculation does not realise the actual positive values and its first column includes zero values. Additional, an F-score *UndefinedMetricWarning* is thrown, setting the value to 0.0, but in the final print out of the Python file this is not always the case.

As a consequence, the following questions appeared: are the original images are realy representative in the same way for the training and testing samples? Is the model architecture, where this effects appeared, a valid one for the given classification task? Are the parameters properly set? Does this only appear because the model shows bias or overfitting? Is there a weakness in the TensorFlow code?

Some questions could be answered, but not all. Not beeing a medical subject matter expert, I am not able to judge if the labelling and the original images are all properly chosen. To have a look to the TensorFlow detailed coding is a project for its own. I have to trust both by now.

But some improvements are made to overcome this training and evaluation results. Unfortunately, because of project deadline the hyperparameter tuning could not be implemented anymore. This has to be a future toDo.

As a conclusion, for the decision about the 'best model', only models can be regarded having a proper training and correct metric values for evaluation and showing bias or overfitting disqualified the model with its configuration already.

Image quality

It would be better to have a more general, good image quality of a specific range, created via machine learning software mainly independent from basic modality and X-ray examinations. This is

not the case with the given dataset and would never be by adding manually created X-ray images from all over the world, by having different tastes and standards in different cultures and countries. Additionally according the modalities and vendors, image quality 'pre-processing' (hardware conditions of tubes and digital detectors and their default handling) and 'post-processing' (software presets) conditions are very different.

To get an image quality that is in general nearly the same for all images needs additional pre-processing coding activities, implemented by a specific machine learning software. This does not exist yet and is a future toDo, but it would improve the performance results of this project models. It is important e.g. for the augmentation technique.

Refinement

Some improvement techniques which are implemented are the following ones:

Augmentation

We are using image data augmentation to decrease the amount of images of our training dataset, and see if there is an improvement to reduce the existing misclassification error. This concept shall improve the performance and the generalisation ability of our network. We want to prevent overfitting. Keras offers already a process to do this, using its *ImageDataGenerator* object class. The original images are rotated, shifted, zoomed, flipped etc. and by doing this, creating new image mutations. So, the range of images for each epoch increases and the propability that the algorithm sees the same images more than once is much reduced.

Generator handling according the given Keras example using .flow() functionality together with the already created pre-processed tensors instances. With such tensors rescaling by /255 has already happened. The .flow() command generates batches of randomly transformed images.

The generator parameters are set according the image properties, having chest images. E.g. huge zooming or rotation would make no sense, but horizontal flip is fine regarding the creation conditions of X-ray images at the modalities.

E.g. zooming is only valid, if we would now the specific image region-of-interest (ROI) for the root cause diagnostic. In other words, if the area where bacteria or virus are assumed shall only be shown in the viewing boundaries (boundary box of the segment) and not the whole thorax anymore. To implement this is a project of its own.

Have in mind, that wrongly set augmentation properties for a task could decrease the training results, means increase the training loss (error costs). E.g. having already an original bad image quality, creating some augmented images on this basic image could make the image quality even worse and it is expected that it would decrease the classification performance. It is not recommended to use all proposed techniques at the same time[17].

In general according augmentation information, have a look to the associated Keras classification article[18].

Optimiser

Better optimiser functions exist: Adam is another adaptive gradient descent algorithm[19]. As described there, Adam combines the advantages of AdaGrad and RMSProp and is very efficient. Less manual hyperparameter tuning for the learning rate schedule is necessary, because Adam

[17] https://gombru.github.io/2017/09/14/data_augmentation/
[18] https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
[19] https://arxiv.org/abs/1412.6980v8

computes adaptive learning rates for each parameter. Adam is the most popular, standard optimiser for neural networks today. First, it is used with its default parameters, then learning rate and decay over each update are tuned. Default:

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False)
```

Transfer learning

Another implemented improvement technique, expected to deliver the best classification performance, is the transfer learning concept. As sense of transfer learning, Goodfellow et al mentioned in their 'Deep Learning' book "Situation where what has been learned in one setting is exploited to improve generalization in another setting."

We don't have a huge dataset for the given pneumonia classification and more generalised, even not for all human age and X-ray orientation categories. The existing dataset is a specific one for a specific task, therefore we need technical help to improve the classification performance having a good, proper learning result, especially thinking at the 'new, available' image examples of the future.

With a pre-trained model, we use its knowledge even from another domain and task (like for image tasks a pre-trained model from the huge ImageNet database including a lot of different object categories), to let our network learn the new problem. This knowledge transfer is possible, because all images share some same features, like e.g. edges, shapes, zooming or rotation. The common features are learned in the hidden layers from the beginning, the more specific task ones are learned at the end of the hidden layer blocks. To improve our specific classification task we can change the network there and let the first part as learned with the ImageNet database. We don't need our network to learn from scratch. We can freeze such hidden layer convolutional subblocks and its outputs are the input of the specific classifier block at the end of the network.

Technically, how to do this is e.g. explained in the same Keras documentation as mentioned in the augmentation text part.

Solutions

Using such improvement methods on the mentioned CNN model architectures, if possible, leads for the 3 different distributions to different results comparing the basic model from scratch and the 'best model' after all training sessions. Having a medical diagnosis classification prediction task, not only the accuracy is important, as mentioned, the number of false negatives as well.

- **Original dataset**

  Remember: Training and testing accuracy of the basic model with rmsprop optimiser: Train: 74.3%, Test: 62.5%

  Best model:

  If it is decided to have a higher prio on the lowest number of false negatives (type 2 error), the ResNet50 transfer model with Adam optimiser and bottleneck features is the best - having only 2 false negatives, but its test accuracy is only 74.2%.

  If the decision is taken to have the model with the highest test accuracy, then the basic, augmented model with Adam optimiser should be chosen: Its value is 87.7%, but having 28 false negatives.

- **V2 70-10-20 dataset**

  Remember: Training and testing accuracy of the basic model with rmsprop optimiser: Train:

75.2%, Test: 63.4%

Best model:

If it is decided to have a higher prio on the lowest number of false negatives (type 2 error), the ResNet50 transfer model with Adam optimiser and bottleneck features is the best - having only 5 false negatives and a test accuracy of 85.91%.

If the decision is taken to have the model with the highest test accuracy, then the InceptionV3 transfer model with Adam optimiser and bottleneck features should be chosen: Its value is 86.34%, but is has detected 15 false negatives.

- **V3 60-20-20 dataset**

  Remember: Training and testing accuracy of the basic model with rmsprop optimiser: Train 77.5%, Test: 63.3%

  Best model:

  If it is decided to have a higher prio on the lowest number of false negatives (type 2 error), the ResNet50 transfer model with Adam optimiser and bottleneck features is the best - having only 5 false negatives and a test accuracy of 84.80%.

  If the decision is taken to have the model with the highest test accuracy, then the Basic Model with Adam optimiser and Augmentation should be chosen: Its value is 89.33%. But is has detected 57 false negatives, which is too much.

As an intermediate insight it has been realised, that using a fine-tuned transfer learning model architecture with augmentation is no general, overall guarantee of having a better classification estimation performance. It even has been one of the models with bad classification performance. But it is unclear now, if the newly activated layers have been chosen suboptimal and if this result can be changed by having optimal hyperparameters.

Another insight has been, that only by using a more complex architecture including more convolutional layers and Adam as optimiser does not mean to have a good or improved estimation performance as well.

So, as a summary, the prediction task must always fit to the model architecture, there is no golden rule that a specific architecure fits to all domains and tasks.

IV. Results

Model Evaluation and Validation

Test accuracy is a common metric regarding the performance of the classifier model, but working on a medical diagnosis issue having severe consequences by chosing a wrong, unsuitable model, this metric value is not enough to compare the different investigated models. It should have been stable on the 3 different dataset distributions as well. At the end for decision, a trade-off has to be made to chose the final model.

It has been decided that the lowest number of false negatives (type 2 error) has the decision criteria. Regarding all 3 distributions always the same model wins:
the **ResNet50 transfer model with Adam optimiser and bottleneck features**.

There are some specific characteristics and parameters:

- The output shape of the InputLayer node is: (None, 224, 224, 3)
- Up to the pooling layer node 'global_average_pooling2d' 23,587,712 non-trainable parameters exists in total.
- The last activation feature map of our ResNet CNN model delivers the bottleneck features. They are the input for the flatten layer and used for our specific, final CNN network top layer.
- This pooling layer node is connected with the next activation node, its output shape is: (Glo (None, 2048)).
- The bottleneck features shape is having a dimension of 1x2048

From the top model part of the network, the

- BatchNormalization node has an output shape: (Batch (None, 2048)) and 8192 parameters
- The final Dense layer has an output shape: (None, 2) and 4098 parameters
- So, params are: `Total: 12,290, trainable: 8,194, Non-trainable: 4,096`

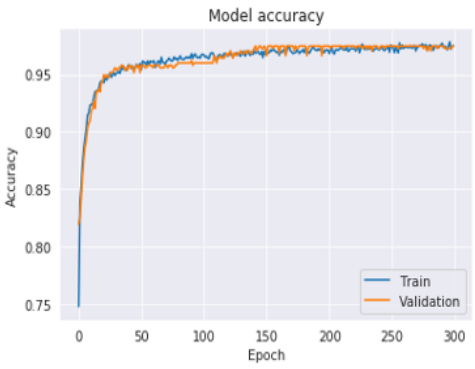
For training configuration: Adam optimiser, weight decay of 1e-6 , reduced learning rate of lr=1e-4

Regarding the metric results in general, the ones of the ResNet50 are better compared to the other models results, especially having a look to the training- and learning curves, the confusion matrix, the ROC-AUC and - thinking at the imbalance - to the Cohen's Kappa values. Details of the metric results of the selected models set is documented in the addendum chapter of the report.

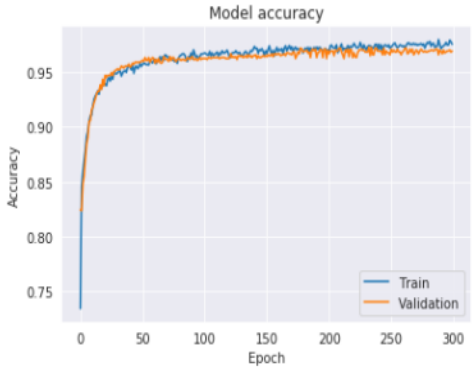
Using an appropriate dataset distribution of training-validation-testing sets (the modified versions V2 70-10-20 or V3-60-20-20) the training- and learning-curves are:

Training accuracy

V2 70-10-20

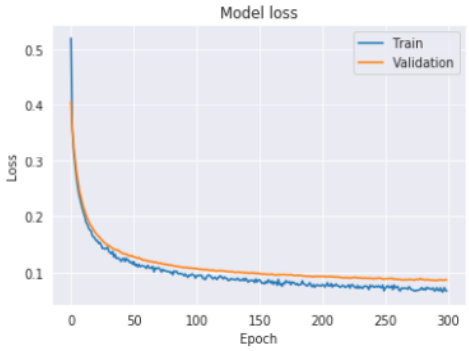


V3 60-20-20

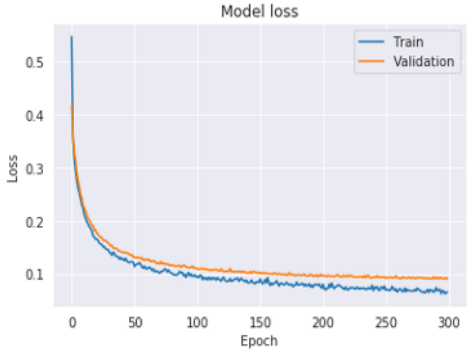


Learning costs

V2 70-10-20



V3 60-20-20



All curves are smooth and nearly the same for training and validation process, which is good.

The **other metrics** of this distributions are fine as well, now the imbalance is even more reduced and the numbers of false negatives are small:

V2 70-10-20

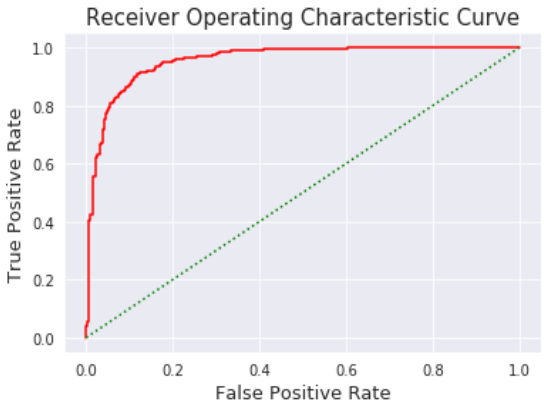
```
Precision: 0.822
Recall: 0.993
F1 score: 0.899
F-beta score: 0.862  with beta=0.5

Cohens kappa: 0.671
Confusion matrix of the test data:
[[265 158]
 [  5 729]]

ROC AUC Pneumonia class prediction:
0.956
```

V3 60-20-20

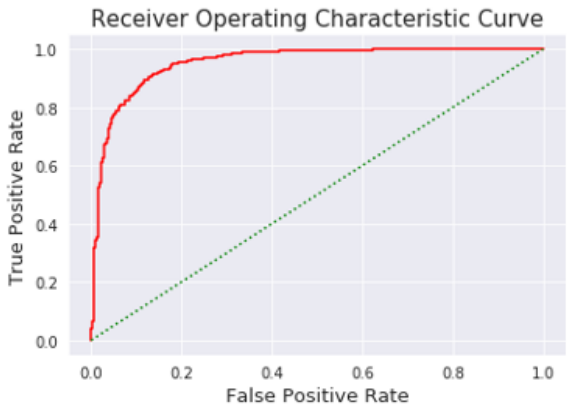
```
Precision: 0.810
Recall: 0.993
F1 score: 0.892
F-beta score: 0.851  with beta=0.5

Cohens kappa: 0.644
Confusion matrix of the test data:
[[257 173]
 [  5 736]]

ROC AUC Pneumonia class prediction:
0.954
```

And the ROC AUC diagrams are:

V2 70-10-20



V3 60-20-20



The investigation results of the ResNet50 model with bottleneck features show small differences depending on the dataset version and its distribution of training-validation-testing sets.

Their test accuracy values are nearly in the direction of the value 86% mentioned by the TÜV representative:

- V2 70-10-20: 85.91%
- V3 60-20-20: 84.80%

But the background information has not been given of how such value has been reached, with what kind of model and model configuration, what kind of dataset was available and what are the results of the other metric concepts (e.g. confusion matrix), therefore such value as benchmark decision is not enough.

Finally, it is expected to have better results for such model after having done a hyperparameter tuning as mentioned in the Improvement chapter.

Justification

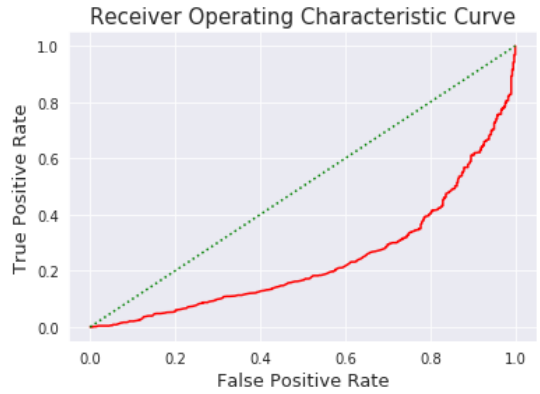
Comparing the final model solution with the benchmark model and some statistics introduced

earlier, we established a significant improvement, so, the solution can be seen as good enough to have solved the problem. Nevertheless as already mentioned, this solution can and should be improved. Here are some results of the benchmark model (basic model with rmsprop optimiser). We show the values for the modified dataset 'V2 70-10-20' as an example. The other ones have issues as well.

V2 70-10-20 dataset

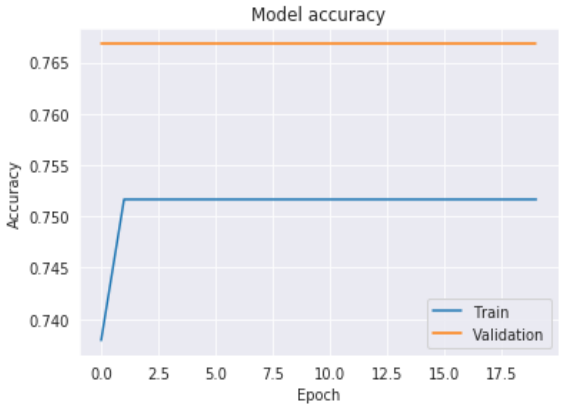
```
Accuracy for training and testing of the basic model with rmsprop optimiser:
Train: 0.752, Test: 0.634

Simple base model with rmsprop optimiser:
Precision: 0.634
Recall: 1.000
F1 score: 0.776
F-beta score: 0.434  with beta=0.5

Cohens kappa: 0.000
Confusion matrix of the test data:
[[  0 423]
 [  0 734]]
```

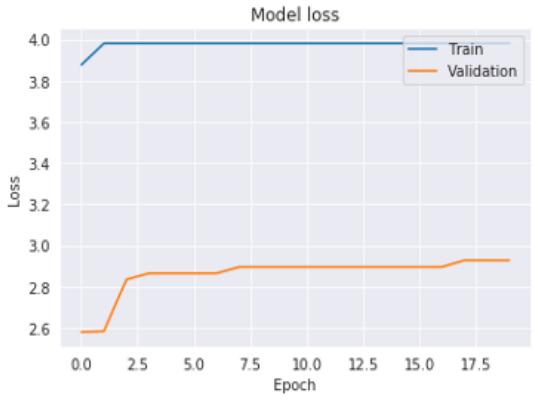
ROC AUC Pneumonia class prediction of basic model with rmsprop optimiser: 0.239



This curve shows, that the models class probabilities are inefficient, even worse than the 50% cutoff of the random events taken by chance.

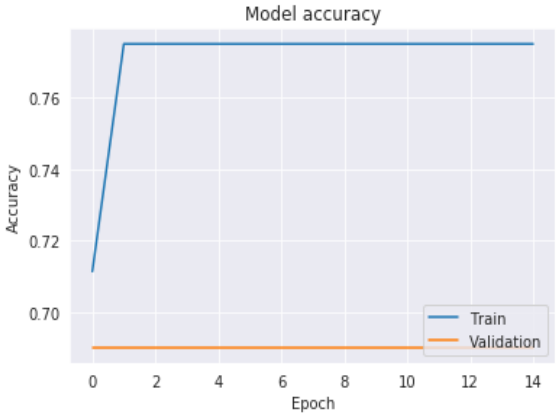
The training- and learning-curves show as well that this model is not suitable for the given task.





The validation curve is much higher compared to the training one, it seems to be a generalisation issue (underfitting) having a biased dataset distribution. The final accuracy value is already reached with the first epoch. So, this model architecture doesn't fit to the classification task.

Having a look to the modified dataset V3 60-20-20, the model accuracy shows overfitting.



The training curve is much higher compared to the validation one, means overfitting exists. Both don't show a real learning during the epochs sequences. The final accuracy value is already reached with the first epoch. So, this model architecture doesn't fit to the classification task.

According Neural Network FAQ[20] "Overfitting is especially dangerous because it can easily lead to predictions that are far beyond the range of the training data with many of the common types of NNs. Overfitting can also produce wild predictions in multilayer perceptrons even with noise-free data." In other words, with overfitting our predictions are not reliable. In the mentioned FAQ several reasons and approaches to avoid overfitting are mentioned.

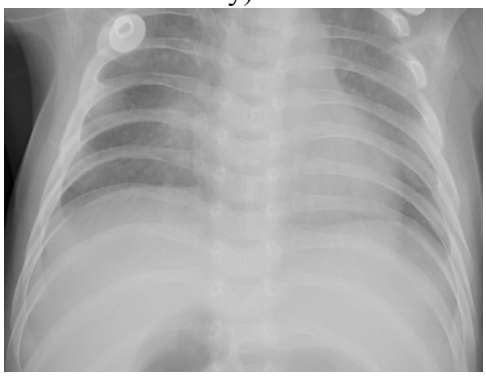
In this project, the usage of dropout layers, changes of decay and learning rate values, augmentation, transfer learning and other model architectures are implemented to avoid over- or underfitting.

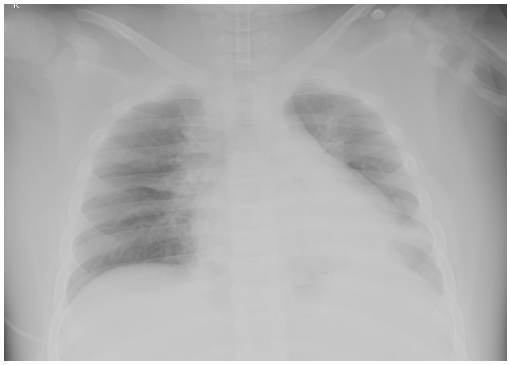
V. Conclusion

Free-Form Visualisation

As realised with visualisation more and more dataset image samples, their image quality differs very much. It is expected that this is an issue making classification predictions much more difficult.

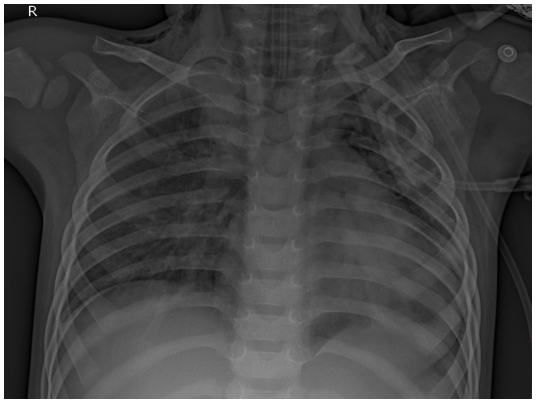
Imagine blurred ones, improperly positioned, like this ones (note: it cannot be visualised in this report properly, it is more sharpened here and to judge about the image quality in general a viewing station is necessary) ...

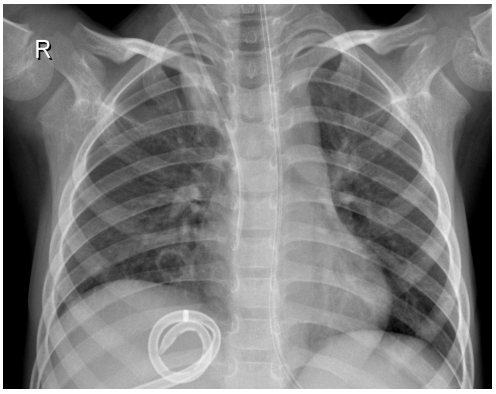




compared to very dark images and/or including other objects like tubes, measuring equipment or implants ...

[20] ftp://ftp.sas.com/pub/neural/FAQ3.html#A_over





Such wide range of image quality aspects and having not enough images from each image quality group, reduces the classification prediction properties of the models. It is an aspect of having an imbalanced dataset as well. And how can we guarantee, that the model does learn of having a pneumonia because of its specific microbes root cause and not because a lot of positive labelled images include (trachea) tubes? In other words, what has been the primary learning aspect, the tube or the virus/bacteria segment?

So, it is expected to have a big improvement according the estimation results by implementing an image quality generalisation algorithm via machine learning concepts as a future toDo.

Reflection

The process used to solve the project task includes the following steps:

- Selection of the domain area and an associated proper, public dataset
- After downloading the dataset, preparing the implementation environment with Python 3.6
- On the dataset, doing some exploration, then creating modified dataset distributions for training-validation-testing splits
- Doing some image pre-processing having valid tensors for the model training process
- First preparing the benchmark classifier model
- Afterwards implementation of additional more complex model architectures
- Implementation of improvements and hyperparameter changes after first trainings on own equipment, resulting in bad predictions and very long training time
- Because of the huge amount of training time: setup of Amazon AWS EC2 instance, used for final model trainings of the 3 dataset distributions

Interesting for me has been the aspect of completely underestimating the training time by usage of few models for only 3 dataset distributions. Another aspect is, how long it takes to find optimal hyperparameters, even not been able to find them for this project, because of improper environment conditions.

A surprising result was the worse results of the fine-tuned transfer learning ResNet50 model. It has been expected to have at least nearly the same result as its associated bottleneck features model or mostly being even better (means higher test accuracy result and good ROC AUC curve value).

Interpreting the training and metrics results has been the most difficult but also most interesting part as well. I was curious about the training results of the model types. As it is described in Christoph

Molnars book 'Interpretable Machine Learning'[21] we as a human want to know what and why a prediction is made as part of our learning behaviour. Also depending on domain and task it is even part of the problem solving process. E.g. for getting an FDA 510(k) approval[22] using a machine-learning application in the US you have to be able to explain and document the 'why and how'.

Finally, it took some time to set the AWS EC2 connection alive, and dealing with the local, unstable internet conditions. At the end it has been disappointing, that I have not found the best model with the best hyperparameters as expected at the beginning, because the project deadline is close and having improper learning conditions. My solution can only be used as a starting point getting a better general solution for such image classification problem solved with deep learning convolutional neural networks.

Improvement

As **further improvement**: Because of project deadline, the applicational algorithm accepting a file path to an unknown, future X-ray .jpg image and determine if it is a pneumonia image or not together with its testing code, has not been implemented yet. This would be possible, e.g. by usage of the skimage package or the OpenCV cv2 implementation.

As an additional **future toDo** to get better classification prediction results of the models, better hyperparameter values must be found, possible by having a better environment for machine learning algorithms. Batch and epoch sizes together with the model parameters initialisation, changing learning rate and early stopping are the aspects to be taken into account. So, hyperparameter tuning from Scikit-Learn with *GridSearchCV* or *RandomizedCV* as an alternative shall be implemented, but it is computational expensive for training having such a lot of parameters for the neural networks. So, we need an appropriate environment to realise this topic.

Furthermore, as a **new insight and future toDo**, for unclear cases having a low estimation result, a deep learning algorithm can be used to improve the image quality as well, if an improper post-processing software preset has been used for the specific patient image or weak X-ray hardware radiation conditions have been chosen. Afterwards, a new classification action could be done automatically. It would ease the diagnostic and communication tasks as well by having a better image quality.
Or as another image quality use case option: already before the training appeared, and as already described in the augmentation refinement text block of this report, all the images could be transfered having a specific general image quality which is nearly the same.

Finally after having implemented this last 2 further improvement topics, only the 2 models having the best classification prediction performance according the decision requirements for this medical domain shall be used for a real application. They should be maintained during the deployment phase. And perhaps, in the future having more images of all ages and orientations, the final model selection would be different. Then more model architectures should be evaluated again.

[21] https://christophm.github.io/interpretable-ml-book/interpretability-importance.html
[22] https://www.fda.gov/; 510(k): mandatory official process of getting an approval to sell and use a medical device in the US

VI. Addendum

Abbreviations

CNN	Convolutional Neural Network
CV	here with machine learning: Cross-Validation (tuning concept)
DICOM	Digital Imaging and Communications in Medicine
dpi	Dots per Inch
FDA	Food & Drug Administration (US regulatory government authoraty)
FN	FalseNegative
FP	FalsePositive
HIV	Human Immunodeficiency Viruses
HIPPA	Health Insurance Portability and Accountability Act
PACS	in the medical domain: Picture Archiving and Communication System
RGB	Red Green Blue
ROI	here with digital image domain: Region-of-Interest
SOP	Service Object Pair
TN	TrueNegative
TP	TruePositive
TÜV	German abbreviation of 'Technischer Überwachungsverein', its English translation is 'Technical Supervisory Association' (regulatory authoraty)
US	United States
WHO	World Health Organisation

Other Model Architectures

- The self-created model having more hidden- using Adam as optimiser at once

```
--- Build model summary of Improved_CNN_Model: ---
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_37 (Conv2D)           (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 112, 112, 16)      0
_________________________________________________________________
dropout_32 (Dropout)         (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 112, 112, 32)      2080
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 111, 111, 32)      4128
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 56, 56, 32)        0
_________________________________________________________________
dropout_33 (Dropout)         (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 56, 56, 64)        8256
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 55, 55, 64)        16448
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 28, 28, 64)        0
_________________________________________________________________
dropout_34 (Dropout)         (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 28, 28, 128)       32896
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 27, 27, 128)       65664
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 14, 14, 128)       0
_________________________________________________________________
dropout_35 (Dropout)         (None, 14, 14, 128)       0
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 14, 14, 256)       131328
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 13, 13, 256)       262400
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 7, 7, 256)         0
_________________________________________________________________
dropout_36 (Dropout)         (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 7, 7, 512)         524800
_________________________________________________________________
conv2d_47 (Conv2D)           (None, 6, 6, 512)         1049088
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 3, 3, 512)         0
_________________________________________________________________
dropout_37 (Dropout)         (None, 3, 3, 512)         0
_________________________________________________________________
global_average_pooling2d_7 ( (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 2)                 1026
=================================================================
Total params: 2,098,322
Trainable params: 2,098,322
Non-trainable params: 0
_________________________________________________________________
```

- The pre-trained ResNet50 CNN model

It starts with …

--- Build model summary of RestNet Transfer_CNN_Model as feature extractor: ---

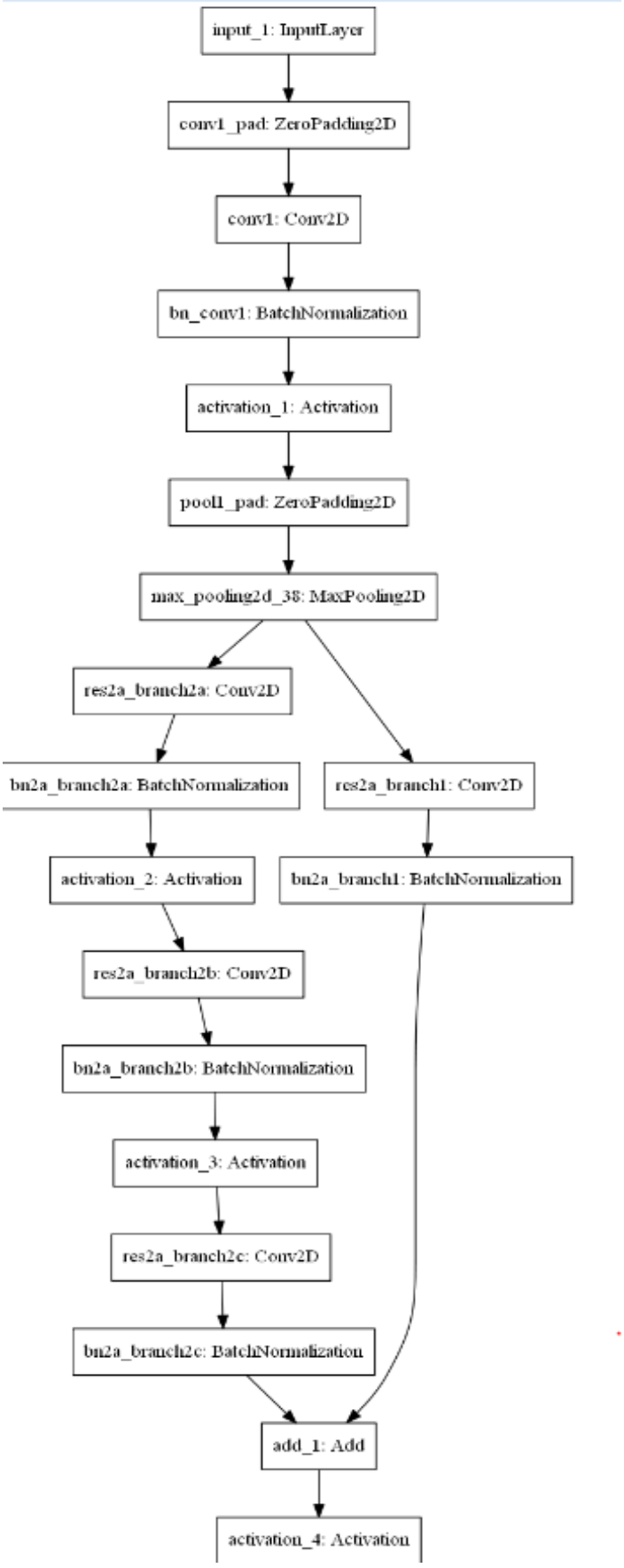
Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	(None, 224, 224, 3)	0	
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_7[0][0]
conv1 (Conv2D)	(None, 112, 112, 64)	9472	conv1_pad[0][0]
bn_conv1 (BatchNormalization)	(None, 112, 112, 64)	256	conv1[0][0]
activation_193 (Activation)	(None, 112, 112, 64)	0	bn_conv1[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	activation_193[0][0]
max_pooling2d_44 (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0][0]
res2a_branch2a (Conv2D)	(None, 56, 56, 64)	4160	max_pooling2d_44[0][0]
bn2a_branch2a (BatchNormalizati	(None, 56, 56, 64)	256	res2a_branch2a[0][0]
activation_194 (Activation)	(None, 56, 56, 64)	0	bn2a_branch2a[0][0]
res2a_branch2b (Conv2D)	(None, 56, 56, 64)	36928	activation_194[0][0]
bn2a_branch2b (BatchNormalizati	(None, 56, 56, 64)	256	res2a_branch2b[0][0]
activation_195 (Activation)	(None, 56, 56, 64)	0	bn2a_branch2b[0][0]
res2a_branch2c (Conv2D)	(None, 56, 56, 256)	16640	activation_195[0][0]
res2a_branch1 (Conv2D)	(None, 56, 56, 256)	16640	max_pooling2d_44[0][0]
bn2a_branch2c (BatchNormalizati	(None, 56, 56, 256)	1024	res2a_branch2c[0][0]
bn2a_branch1 (BatchNormalizatio	(None, 56, 56, 256)	1024	res2a_branch1[0][0]
add_33 (Add)	(None, 56, 56, 256)	0	bn2a_branch2c[0][0] bn2a_branch1[0][0]
activation_196 (Activation)	(None, 56, 56, 256)	0	add_33[0][0]
res2b_branch2a (Conv2D)	(None, 56, 56, 64)	16448	activation_196[0][0]
bn2b_branch2a (BatchNormalizati	(None, 56, 56, 64)	256	res2b_branch2a[0][0]
activation_197 (Activation)	(None, 56, 56, 64)	0	bn2b_branch2a[0][0]
res2b_branch2b (Conv2D)	(None, 56, 56, 64)	36928	activation_197[0][0]
bn2b_branch2b (BatchNormalizati	(None, 56, 56, 64)	256	res2b_branch2b[0][0]
activation_198 (Activation)	(None, 56, 56, 64)	0	bn2b_branch2b[0][0]

and ends up with ...

activation_237 (Activation)	(None, 7, 7, 512)	0	bn5b_branch2b[0][0]
res5b_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_237[0][0]
bn5b_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5b_branch2c[0][0]
add_47 (Add)	(None, 7, 7, 2048)	0	bn5b_branch2c[0][0] activation_235[0][0]
activation_238 (Activation)	(None, 7, 7, 2048)	0	add_47[0][0]
res5c_branch2a (Conv2D)	(None, 7, 7, 512)	1049088	activation_238[0][0]
bn5c_branch2a (BatchNormalizati	(None, 7, 7, 512)	2048	res5c_branch2a[0][0]
activation_239 (Activation)	(None, 7, 7, 512)	0	bn5c_branch2a[0][0]
res5c_branch2b (Conv2D)	(None, 7, 7, 512)	2359808	activation_239[0][0]
bn5c_branch2b (BatchNormalizati	(None, 7, 7, 512)	2048	res5c_branch2b[0][0]
activation_240 (Activation)	(None, 7, 7, 512)	0	bn5c_branch2b[0][0]
res5c_branch2c (Conv2D)	(None, 7, 7, 2048)	1050624	activation_240[0][0]
bn5c_branch2c (BatchNormalizati	(None, 7, 7, 2048)	8192	res5c_branch2c[0][0]
add_48 (Add)	(None, 7, 7, 2048)	0	bn5c_branch2c[0][0] activation_238[0][0]
activation_241 (Activation)	(None, 7, 7, 2048)	0	add_48[0][0]
global_average_pooling2d_11 (Gl	(None, 2048)	0	activation_241[0][0]

Total params: 23,587,712
Trainable params: 0
Non-trainable params: 23,587,712

This ResNet architecture starts like this, and in principle, using parallel calculation branches heavily:



- The pre-trained InceptionV3 CNN model

It starts with …

```
--- Build model summary of InceptionV3 Transfer_CNN_Model as feature extractor: ---
```

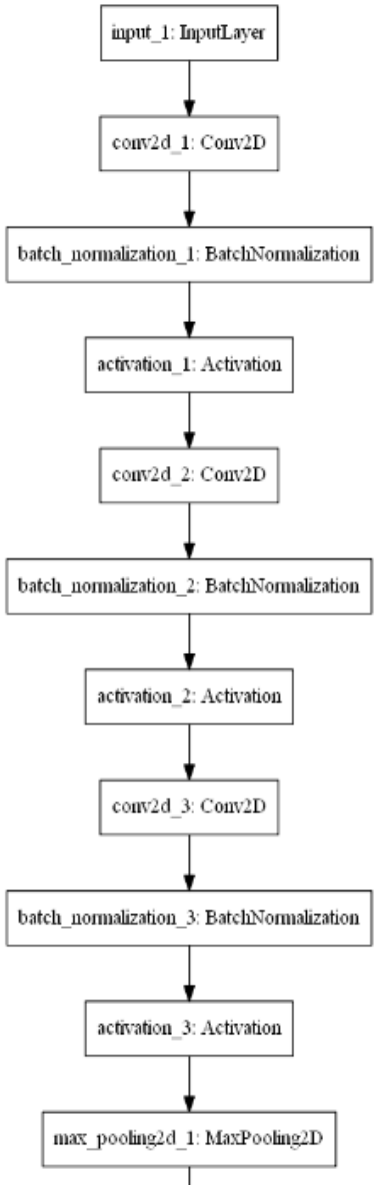
Layer (type)	Output Shape	Param #	Connected to
input_9 (InputLayer)	(None, 224, 224, 3)	0	
conv2d_142 (Conv2D)	(None, 111, 111, 32)	864	input_9[0][0]
batch_normalization_99 (BatchNo	(None, 111, 111, 32)	96	conv2d_142[0][0]
activation_242 (Activation)	(None, 111, 111, 32)	0	batch_normalization_99[0][0]
conv2d_143 (Conv2D)	(None, 109, 109, 32)	9216	activation_242[0][0]
batch_normalization_100 (BatchN	(None, 109, 109, 32)	96	conv2d_143[0][0]
activation_243 (Activation)	(None, 109, 109, 32)	0	batch_normalization_100[0][0]
conv2d_144 (Conv2D)	(None, 109, 109, 64)	18432	activation_243[0][0]
batch_normalization_101 (BatchN	(None, 109, 109, 64)	192	conv2d_144[0][0]
activation_244 (Activation)	(None, 109, 109, 64)	0	batch_normalization_101[0][0]
max_pooling2d_45 (MaxPooling2D)	(None, 54, 54, 64)	0	activation_244[0][0]
conv2d_145 (Conv2D)	(None, 54, 54, 80)	5120	max_pooling2d_45[0][0]
batch_normalization_102 (BatchN	(None, 54, 54, 80)	240	conv2d_145[0][0]
activation_245 (Activation)	(None, 54, 54, 80)	0	batch_normalization_102[0][0]
conv2d_146 (Conv2D)	(None, 52, 52, 192)	138240	activation_245[0][0]
batch_normalization_103 (BatchN	(None, 52, 52, 192)	576	conv2d_146[0][0]
activation_246 (Activation)	(None, 52, 52, 192)	0	batch_normalization_103[0][0]
max_pooling2d_46 (MaxPooling2D)	(None, 25, 25, 192)	0	activation_246[0][0]
conv2d_150 (Conv2D)	(None, 25, 25, 64)	12288	max_pooling2d_46[0][0]

and ends up with …

conv2d_229 (Conv2D)	(None, 5, 5, 384)	442368	activation_328[0][0]
conv2d_230 (Conv2D)	(None, 5, 5, 384)	442368	activation_328[0][0]
conv2d_233 (Conv2D)	(None, 5, 5, 384)	442368	activation_332[0][0]
conv2d_234 (Conv2D)	(None, 5, 5, 384)	442368	activation_332[0][0]
average_pooling2d_18 (AveragePo	(None, 5, 5, 2048)	0	mixed9[0][0]
conv2d_227 (Conv2D)	(None, 5, 5, 320)	655360	mixed9[0][0]
batch_normalization_186 (BatchN	(None, 5, 5, 384)	1152	conv2d_229[0][0]
batch_normalization_187 (BatchN	(None, 5, 5, 384)	1152	conv2d_230[0][0]
batch_normalization_190 (BatchN	(None, 5, 5, 384)	1152	conv2d_233[0][0]
batch_normalization_191 (BatchN	(None, 5, 5, 384)	1152	conv2d_234[0][0]
conv2d_235 (Conv2D)	(None, 5, 5, 192)	393216	average_pooling2d_18[0][0]
batch_normalization_184 (BatchN	(None, 5, 5, 320)	960	conv2d_227[0][0]
activation_329 (Activation)	(None, 5, 5, 384)	0	batch_normalization_186[0][0]
activation_330 (Activation)	(None, 5, 5, 384)	0	batch_normalization_187[0][0]
activation_333 (Activation)	(None, 5, 5, 384)	0	batch_normalization_190[0][0]
activation_334 (Activation)	(None, 5, 5, 384)	0	batch_normalization_191[0][0]
batch_normalization_192 (BatchN	(None, 5, 5, 192)	576	conv2d_235[0][0]
activation_327 (Activation)	(None, 5, 5, 320)	0	batch_normalization_184[0][0]
mixed9_1 (Concatenate)	(None, 5, 5, 768)	0	activation_329[0][0] activation_330[0][0]
concatenate_4 (Concatenate)	(None, 5, 5, 768)	0	activation_333[0][0] activation_334[0][0]
activation_335 (Activation)	(None, 5, 5, 192)	0	batch_normalization_192[0][0]
mixed10 (Concatenate)	(None, 5, 5, 2048)	0	activation_327[0][0] mixed9_1[0][0] concatenate_4[0][0] activation_335[0][0]
global_average_pooling2d_12 (Gl	(None, 2048)	0	mixed10[0][0]

Total params: 21,802,784
Trainable params: 0
Non-trainable params: 21,802,784

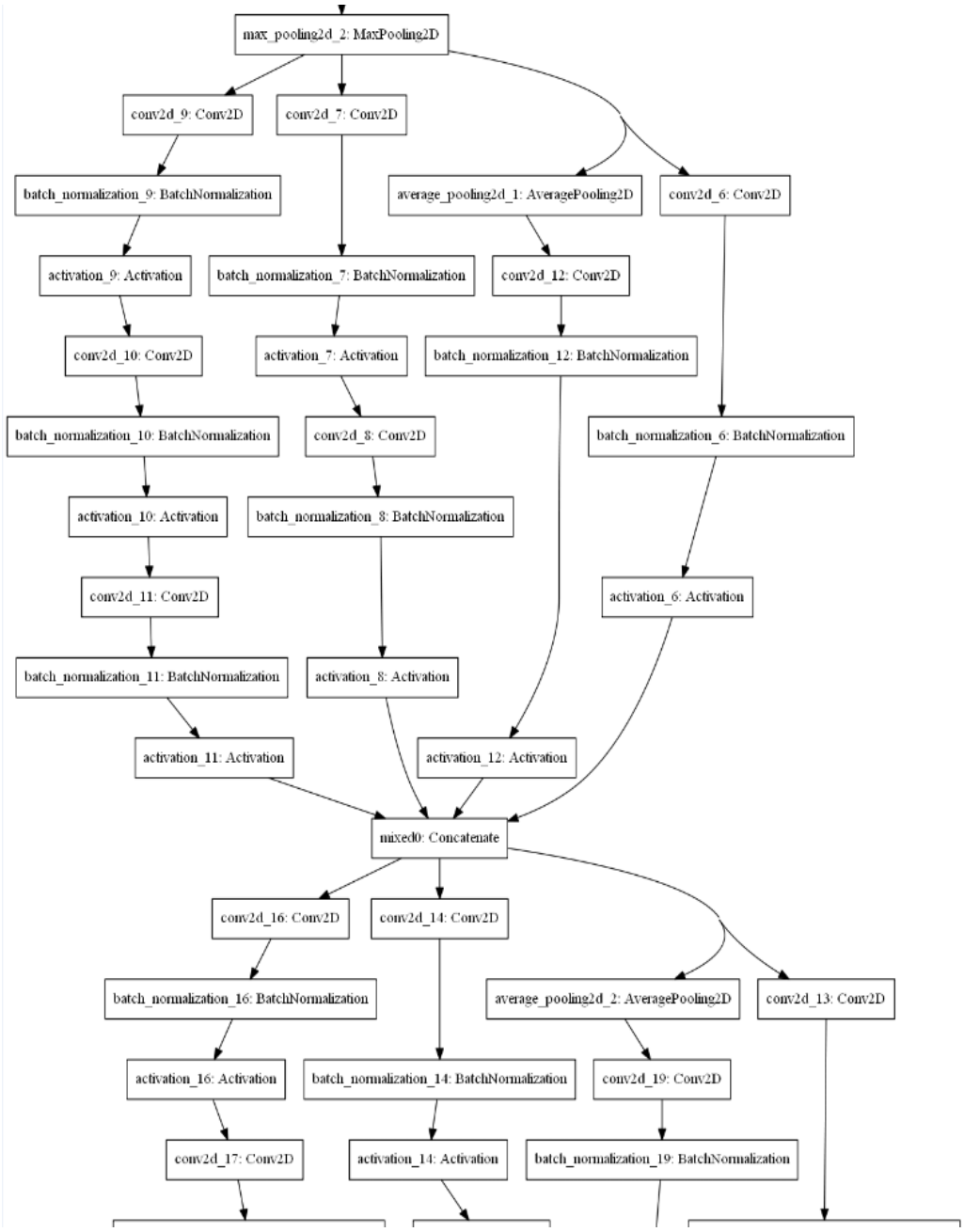
This ResNet architecture starts like this, in a simple way



but then is is a very complex architecture using a lot of parallel layer subblocks for calculations;

subblocks are concatenated and

much more subblocks like the following one exist before it end up in a mixed concatenated node followed by the flatten GlobalAveragePooling2D layer.

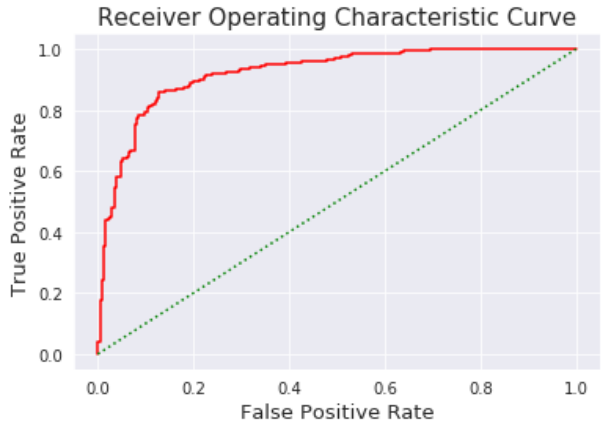

max_pooling2d_2: MaxPooling2D
conv2d_9: Conv2D
conv2d_7: Conv2D
batch_normalization_9: BatchNormalization
average_pooling2d_1: AveragePooling2D
conv2d_6: Conv2D
activation_9: Activation
batch_normalization_7: BatchNormalization
conv2d_12: Conv2D
conv2d_10: Conv2D
activation_7: Activation
batch_normalization_12: BatchNormalization
batch_normalization_10: BatchNormalization
conv2d_8: Conv2D
batch_normalization_6: BatchNormalization
activation_10: Activation
batch_normalization_8: BatchNormalization
conv2d_11: Conv2D
activation_6: Activation
batch_normalization_11: BatchNormalization
activation_8: Activation
activation_11: Activation
activation_12: Activation
mixed0: Concatenate
conv2d_16: Conv2D
conv2d_14: Conv2D
batch_normalization_16: BatchNormalization
average_pooling2d_2: AveragePooling2D
conv2d_13: Conv2D
activation_16: Activation
batch_normalization_14: BatchNormalization
conv2d_19: Conv2D
conv2d_17: Conv2D
activation_14: Activation
batch_normalization_19: BatchNormalization

Detailed Metric Results of the Selected Models Set

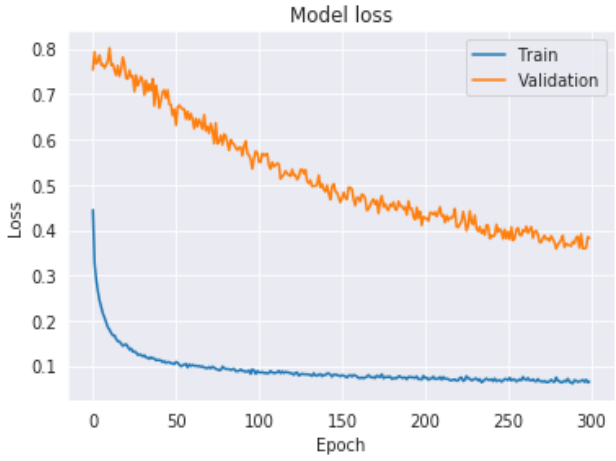
Original Dataset

Prio: small FN:
If it is decided to have a higher prio on the lowest number of false negatives (type 2 error), the ResNet50 transfer model with Adam optimiser and bottleneck features - having only 2 false negatives, but its test accuracy is only 74.2%.

```
ResNet50 transfer model with Adam optimiser and bottleneck features:
Precision: 0.709
Recall: 0.995
F1 score: 0.828
F-beta score: 0.730  with beta=0.5

Cohens kappa: 0.364
Confusion matrix of the test data:
[[ 75 159]
 [  2 388]]

ROC AUC Pneumonia class prediction of ResNet50 transfer model with Adam
optimiser and bottleneck features: 0.921
```

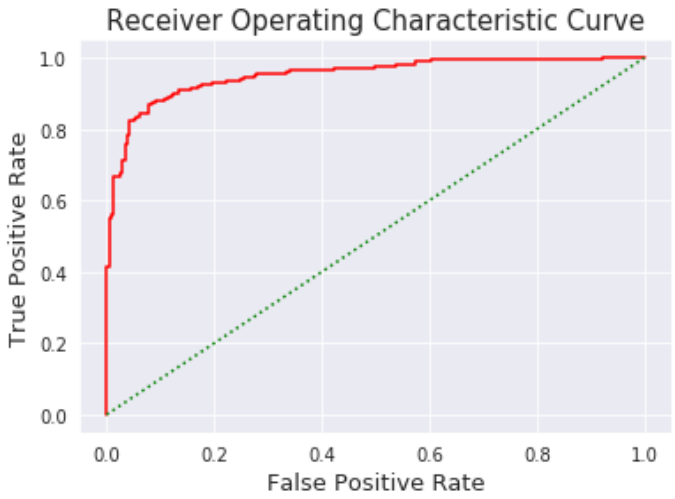


ResNet50 transfer model with Adam optimiser and bottleneck features:

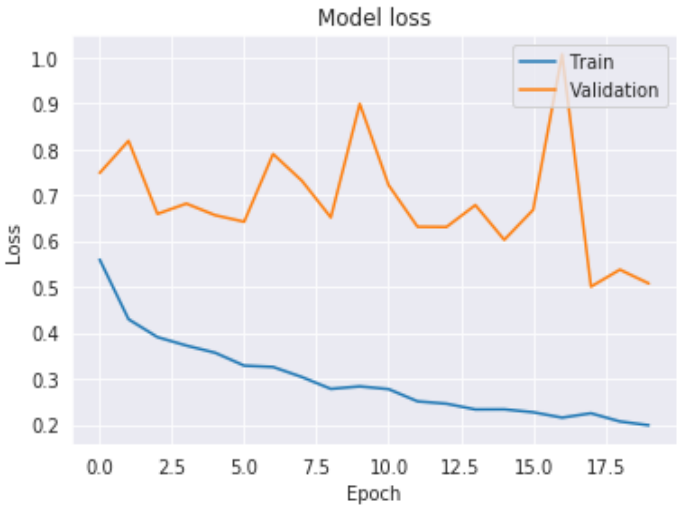


Prio: highest test-accuracy:
If the decision is taken to have the model with the highest test accuracy, then the basic, augmented model with Adam optimiser should be chosen: Its value is 87.7%.

```
Basic, augmented model with Adam optimiser:
Precision: 0.881
Recall: 0.928
F1 score: 0.904
F-beta score: 0.876  with beta=0.5

Cohens kappa: 0.732
Confusion matrix of the test data:
[[185  49]
 [ 28 362]]

ROC AUC Pneumonia class prediction of basic, augmented model with Adam
optimiser: 0.951
```



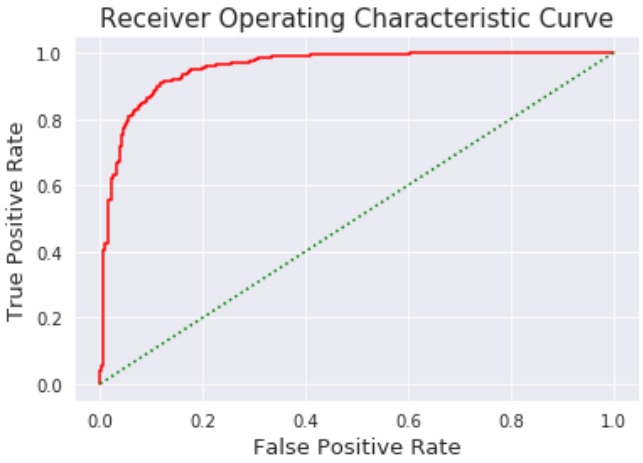
Basic, augmented model with Adam optimiser:



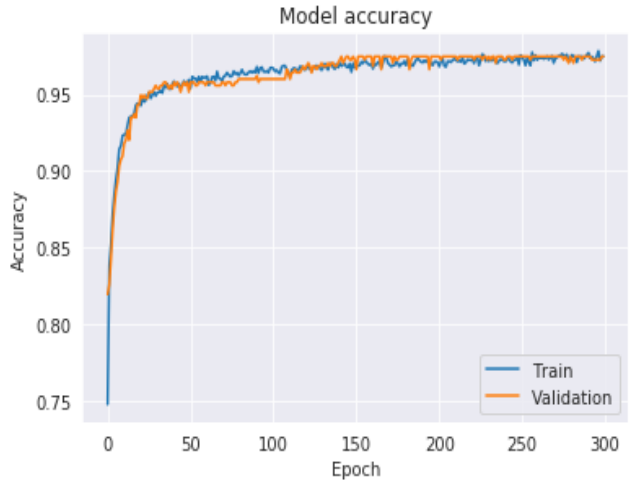
V2 70-10-20 dataset

Prio: small FN:
If it is decided to have a higher prio on the lowest number of false negatives (type 2 error), the ResNet50 transfer model with Adam optimiser and bottleneck features - having only 5 false negatives and a test accuracy of 85.91%.

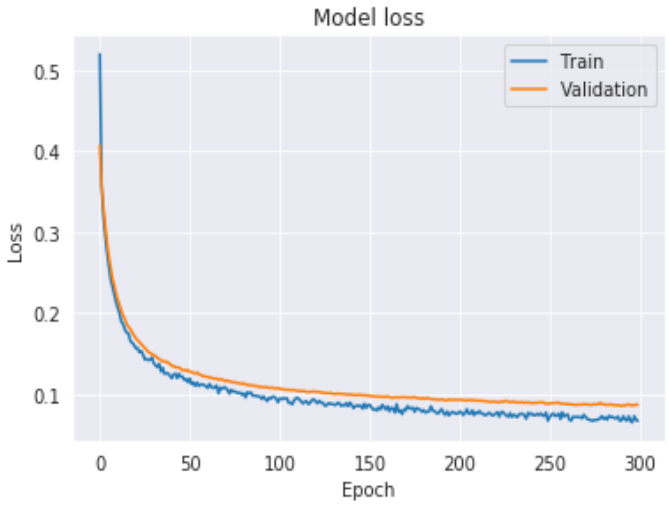
```
ResNet50 transfer model with Adam optimiser and bottleneck features:
Precision: 0.822
Recall: 0.993
F1 score: 0.899
F-beta score: 0.862  with beta=0.5

Cohens kappa: 0.671
Confusion matrix of the test data:
[[265 158]
 [  5 729]]

ROC AUC Pneumonia class prediction of ResNet50 transfer model with Adam
optimiser and bottleneck features: 0.956
```



ResNet50 transfer model with Adam optimiser and bottleneck features:

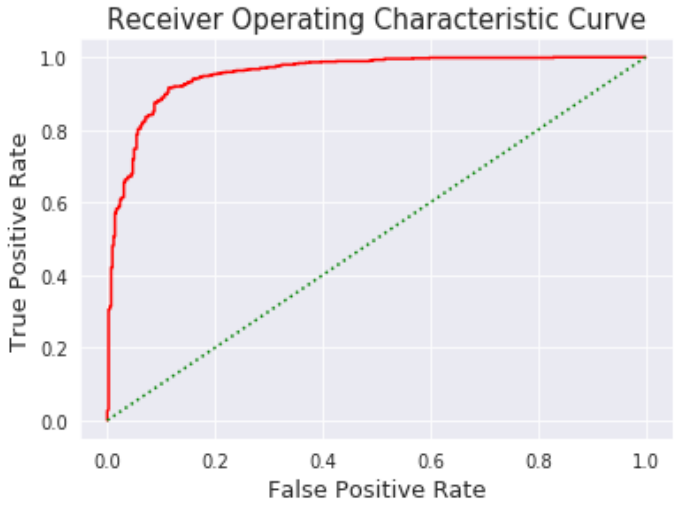


```
ResNet50 transfer model with Adam optimiser and bottleneck features:
```

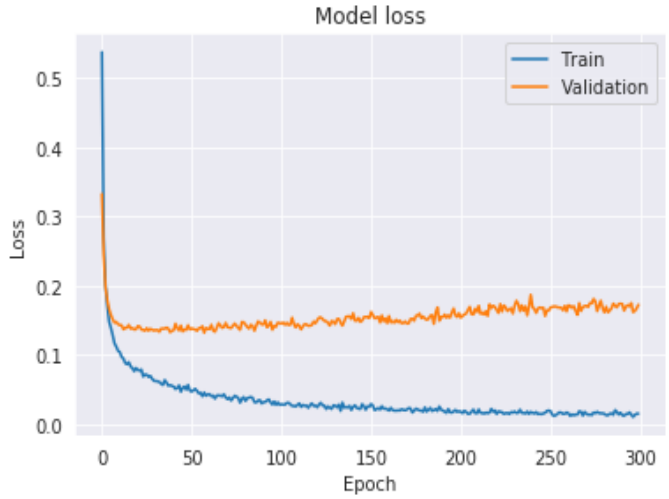


Prio: highest test-accuracy:
If the decision is taken to have the model with the highest test accuracy, then the InceptionV3 transfer model with Adam optimiser and bottleneck features should be chosen: Its value is 86.34%. But is has detected 15 false negatives.

```
InceptionV3 transfer model with Adam optimiser and bottleneck features:
Precision: 0.834
Recall: 0.980
F1 score: 0.901
F-beta score: 0.865  with beta=0.5

Cohens kappa: 0.685
Confusion matrix of the test data:
[[280 143]
 [ 15 719]]

ROC AUC Pneumonia class prediction of InceptionV3 transfer model with Adam
optimiser: 0.955
```



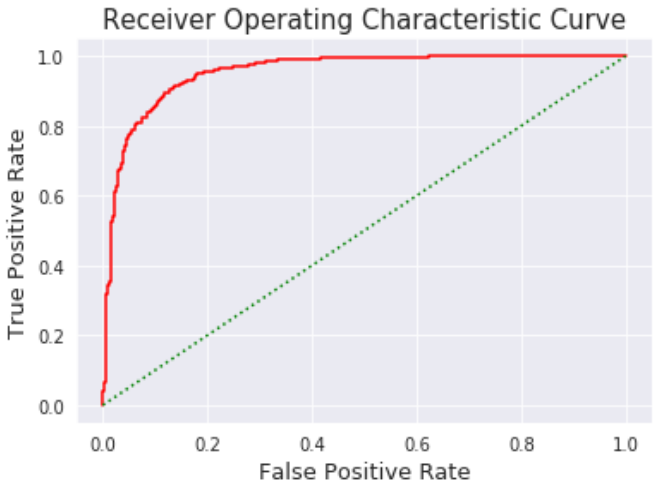
```
InceptionV3 transfer model with Adam optimiser and bottleneck features:
```



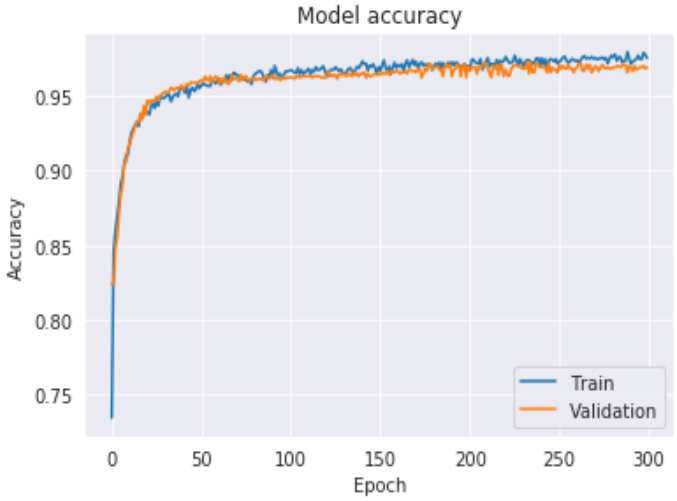
V3 60-20-20

Prio: small FN:
If it is decided to have a higher prio on the lowest number of false negatives (type 2 error), the ResNet50 transfer model with Adam optimiser and bottleneck features - having only 5 false negatives and a test accuracy of 84.80%.

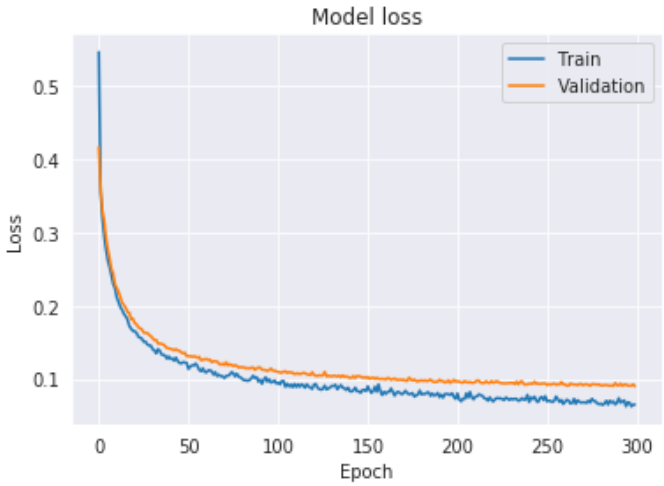
```
ResNet50 transfer model with Adam optimiser and bottleneck features:
Precision: 0.810
Recall: 0.993
F1 score: 0.892
F-beta score: 0.851  with beta=0.5

Cohens kappa: 0.644
Confusion matrix of the test data:
[[257 173]
 [  5 736]]

ROC AUC Pneumonia class prediction of ResNet50 transfer model with Adam
optimiser and bottleneck features: 0.954
```



ResNet50 transfer model with Adam optimiser and bottleneck features:



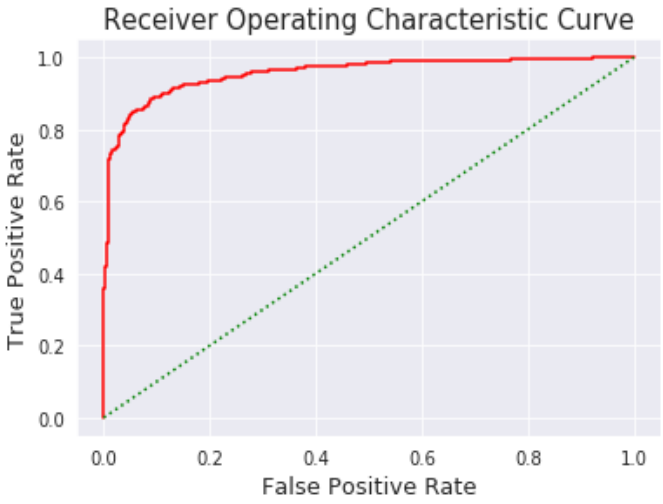
ResNet50 transfer model with Adam optimiser and bottleneck features:



Prio: highest test-accuracy:
If the decision is taken to have the model with the highest test accuracy, then the Basic Model with Adam optimiser and Augmentation should be chosen: Its value is 89.33%. But is has detected 57 false negatives, which is too much.

```
Basic, augmented model with Adam optimiser:
Precision: 0.910
Recall: 0.923
F1 score: 0.916
F-beta score: 0.893  with beta=0.5

Cohens kappa: 0.769
Confusion matrix of the test data:
[[362  68]
 [ 57 684]]
```

ROC AUC Pneumonia class prediction of basic, augmented model with Adam optimiser: 0.957



Basic, augmented model with Adam optimiser:

Benchmark model: basic model with rmsprop optimiser

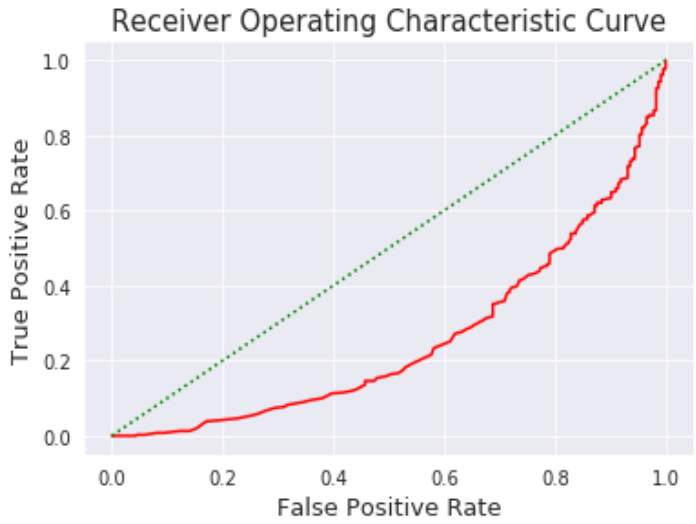
Original Dataset

Accuracy for training and testing of the basic model with rmsprop optimiser:
Train: 0.743, Test: 0.625

Simple base model with rmsprop optimiser:
Precision: 0.625
Recall: 1.000
F1 score: 0.769
F-beta score: 0.422 with beta=0.5

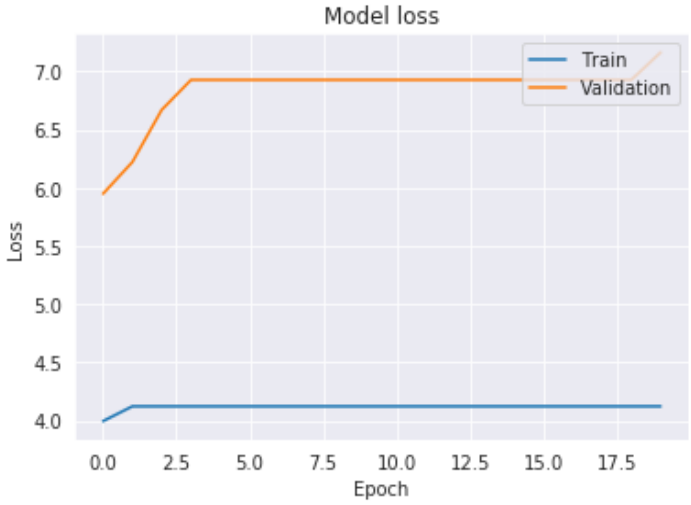
Cohens kappa: 0.000
Confusion matrix of the test data:
[[0 234]
 [0 390]]

```
Thrown is: UndefinedMetricWarning: F-score is ill-defined and being set to 0.0
in labels with no predicted samples.

ROC AUC Pneumonia class prediction of basic model with rmsprop optimiser: 0.257
```



The ROC AUC curve is a worse one compared to the random, green dotted line which appeared by chance. This model is inefficient for estimation of class probabilities. This can be realised with the loss curve as well.

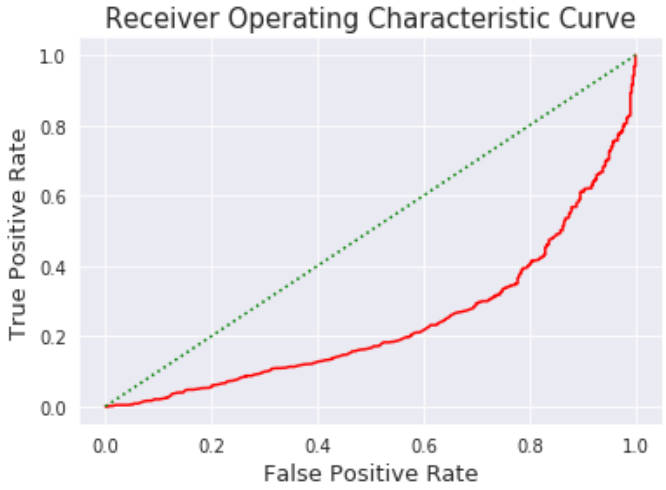
```
Simple base model with rmsprop optimiser:
```



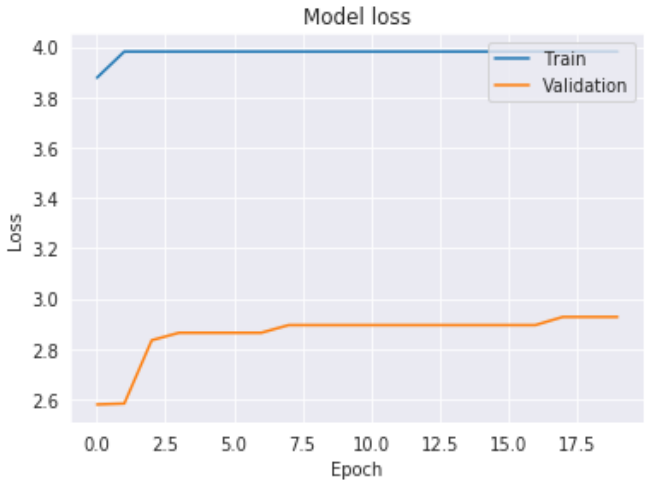
V2 70-10-20 dataset

```
Accuracy for training and testing of the basic model with rmsprop optimiser:
Train: 0.752, Test: 0.634

Simple base model with rmsprop optimiser:
Precision: 0.634
Recall: 1.000
F1 score: 0.776
F-beta score: 0.434  with beta=0.5

Cohens kappa: 0.000
```

```
Confusion matrix of the test data:
[[  0 423]
 [  0 734]]

ROC AUC Pneumonia class prediction of basic model with rmsprop optimiser: 0.239
```



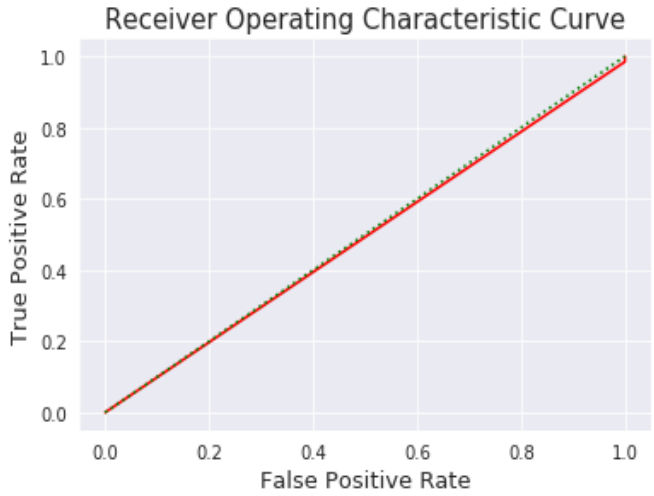
Simple base model with rmsprop optimiser:



V3 60-20-20

```
Accuracy for training and testing of the basic model with rmsprop optimiser:
Train: 0.775, Test: 0.633

Simple base model with rmsprop optimiser:
Precision: 0.633
Recall: 1.000
F1 score: 0.775
F-beta score: 0.432  with beta=0.5

Cohens kappa: 0.000
Confusion matrix of the test data:
[[  0 430]
 [  0 741]]
```

ROC AUC Pneumonia class prediction of basic model with rmsprop optimiser: 0.493



Simple base model with rmsprop optimiser: