ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Deep Learning for Smartphone-based Cough Detection

Master's Thesis

Michael Bachmann

michabact@student.ethz.ch

Information Management Group & Distributed Computing Group
MTEC & ITET Departments
ETH Zürich

| | |
|---|---|
| Supervisor: | Dr. Filipe Da Conceicao Barata |
| Advisor: | David Cleres |
| Co-Supervisor: | Prof. Dr. Roger Wattenhofer |

December 20, 2021

# Acknowledgements

I would like to thank Dr. Filipe Da Conceicao Barata for his support, supervision, guidance and countless helpful concept explanations and clarifications. Also, I would like to thank David Cleres for his well structured and documented code base, his supervision and neverending patience in helping to get the code running. Furthermore, I would like to thank Prof. Dr. Roger Wattenhofer for co-supervising the Thesis. Last but not least I would like to thank the Information Management Group for this great opportunity and I would like to thank the MTEC and ETHZ IT services team for helping with the infrastructure.

# Abstract

Coughing is the most common reason people seek medical advice and is a symptom of numerous diseases that can end severely if not monitored continuously. Finding an automated smartphone-based cough detection and counting application could potentially quite affordably save lives and reduce health cost by automatically recognizing a decline in condition in an early stage. This would, with a high chance, prevent a more critical course of disease as treatment could be initiated in time. We aimed to improve such a detector from prior work using residual convolutional neural networks and by examining the postprocessing stage. For that, audio from 94 adults with asthma was used (mean 43 years; standard deviation 16 years; female: 57%; male: 43%) with a total of 30'304 cough sounds. The mean cough difference per day between human-annotated coughs and counted coughs by the automated detector was 0 coughs with a 95% confidence interval of -11.97 and 11.98 coughs or with other thresholds 0.06 coughs with a 95% confidence interval of –11.62 and 11.75. These results are slightly better than Barata et al. 2020's cough detector and were achieved with a new architecture using residual convolutional neural networks. Additionally, on a different dataset with data from 10 covid-19 patients with a total of 3888 cough sounds the postprocessing stage was assessed and improved by 0.016 MCC (+2.1%) which implicates that there is even more potential for improvement of the cough detector.

# Contents

Chapter 1

# Introduction

Coughing is the most common reason why people seek medical advice [1] [2] [3]. The cause of this symptom may be something like just a common cold, but it can also be a sign for numerous causes that are much more severe. To name a few, coughing can be an indicator for chronic obstructive pulmonary disease (COPD), asthma, tuberculosis, gastro-oesophageal reflux, chronic bronchitis, cystic fibrosis and even for some heart diseases like congestive heart failure [4] [5] [6]. Thus monitoring and analysis of coughing behavior have the potential to be a critical instrument in detecting a change in disease progression. This in turn may prevent a more severe development. The question asked however should be how does one monitor and keep a record of cough as a symptom? An intuitive approach is measuring the symptom severity by monitoring the frequency of coughs or cough clusters. Unfortunately, the self-assessment and self-record of the cough frequency are quite unreliable and also inconvenient for the patient [7]. As humans are much more susceptible to errors than machines, an automatic machine learning-based cough detection device could be an alternative with a lot of potential. This device would have to be ubiquitous on a daily basis and possess the required hard- and software. Nowadays, with the increasing computation power, increasing battery life, diverse audio hardware, flexible application interface and already societal established ubiquity, a smartphone seems like a great choice for this task.

A Problem with an audio-based detection device in daily life is all the background noise and the challenge of the discrimination between different individuals on an audio recording. To mitigate this, the application could monitor only coughs that happen during bedtime, as the recording will most likely only contain audio from the individual in question plus possibly a partner. Additionally, there is a high chance the recording will have significantly less background noise compared to a daytime recording.

Quite some work has already been published by various researchers about cough detection [3], however, there is not yet one that you would call the gold standard of cough detectors [1] [3] and neither one that uses residual neural networks specifically. One difficulty when trying to create a good cough detector is the rareness of the occurrence of a cough when trying to collect data and

build a well representative model with it. This means there is an extremely strong natural class imbalance which can be an obstacle when trying to build a good classifier. Another difficulty is that the computing power and the power consumption of the detection algorithm are limited to the specifications of the ubiquitous device, which in our case would be the smartphone. The advantage of a smartphone compared to a device specifically made for this task is that among the target society basically everyone has a smartphone available and would only have to acquire the software via an App-download instead of acquiring and taking along an additional device.

Such a cough detector, as described above, does exist and in this thesis, the research goal was to accomplish better Results than said [8] detector with the newer TensorFlow framework version two. Residual neural networks are reported to show better results in comparison to conventional neural networks [9]. This applies especially to deeper networks as with residual connections, crucial information from earlier layers can be passed to subsequent layers virtually unchanged and hence make its way through towards the output. Therefore we aimed to investigate to what extent neural network-based cough detection can be improved by employing residual neural networks. Additionally, we considered other non-architectural approaches with the postprocessing stage as the second main target and thus aimed to investigate to what extent machine learning-based postprocessing can improve predicted cough counts.

Chapter 2

# Related Work

This thesis is based on Dr. Filipe Da Conceicao Barata's Work [8] [3]. Said work includes extensive research and a cough detection model that achieved a Matthews correlation coefficient of 92% regarding pure classification. The count difference per day between automated and human-annotated coughs was a mean of –0.1 coughs with a 95% confidence interval of –12.11 and 11.91 coughs. The data that was used was collected in this study [8]. As for the training data, 650ms cough and non-cough (noise) audio samples were isolated and centered. The computed Mel spectrogram of the audio samples was labeled and the noise samples were split into five different datasets for ensemble learning with five times the same cough sample dataset but five different noise sample datasets. The model architecture was a convolutional neural network (CNN) classifier with a softmax function at the end to output the desired probabilities. Said approach focuses on the weight optimization process of the neural network instead of focusing on handcrafting the most significant features, unlike most other prior work in this field. Also, this study did longitudinal overnight testing, which is a much more representative respectively generalizable evaluation compared to an evaluation on specifically isolated cough and noise samples.

Chapter 3

# Theoretical Background

## 3.1 Melspectrogram

The discrete Fourier transform (DFT) $\hat{x}(k)$ see equation 3.1 of audio signals has been known to be a popular feature in audio analysis with which you can extract the occurring frequencies of an entire audio signal.

$$\hat{x}(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-i2\pi n \frac{k}{N}} \tag{3.1}$$

However, if a temporal dependency is required, which in most use cases applies, the short-term Fourier transform (STFT) $S(m, k)$ see equation 3.2 is better suited.

$$S(m, k) = \sum_{n=0}^{N-1} x(n + mH) \cdot w(n) \cdot e^{-i2\pi n \frac{k}{N}} \tag{3.2}$$

The STFT basically is a windowed version of the DFT where the vertical result vectors of the single windows are concatenated horizontally to a two-dimensional matrix with complex entries. In equation 3.2 $w(n)$ denotes the Hann-window, H the hop-size and m the window-index. If you take the squared magnitude of the STFT $S(m, k)$ you get the spectrogram $Y(m, k)$ of the signal which is a two-dimensional representation of a one-dimensional signal with only real numbers [10] see equation 3.3. This can now be treated in a similar fashion as ordinary images in automated classification tasks.

$$Y(m, k) = |S(m, k)|^2 \tag{3.3}$$

To get the Mel spectrogram, the frequency-axis and therefore also all the values yet need to be skewed according to the Mel scale. See equation 3.4 where m stands for the frequency in Mel and f the frequency in Hz [11].

$$m = 2595 \cdot log_{10}(1 + \frac{f}{700}) \tag{3.4}$$

The Mel scale treats sounds linearly not in terms of frequency but in terms of human perception which is known to give good results in audio classification.

## 3.2 Depthwise Separable 2D Convolution

Convolution layers in general, require quite a lot of computation power. Depthwise separable 2D convolution (sepconv) is a convolution method that slightly tries to mitigate this problem, sometimes for the cost of predictive performance. To explain the difference between ordinary 2D convolution (ordconv) and sepconv let us assume we want to perform an m×n convolution with o output channels on an input with i input channels. In ordinary 2D convolution, the filter size of one filter is (m, n, i). Its resulting convolution returns one channel per filter, which means we need o filters for our example scenario. The more filters you use, the more output channels you obtain. Convolutions with such large filters are computationally quite expensive, leading to inefficiency where a lot of output channels are required. In contrast, sepconv involves two separate convolution stages. The filter sizes of the first convolution stage are (m, n, 1), and exactly *1* of those smaller 1D filters per input channel is applied and optimized in this stage. Up so far, the sepconv has done as much computation as the normal convolution would have if it had only one output channel. The second stage of this convolution intuitively is just an aggregation of the obtained intermediate results into one channel in o different ways. The filter size for the second stage is (1, 1, i). For every desired output channel, exactly one such 1×1 filter is applied and optimized which returns o output channels. 1×1 convolution is quite efficient computationwise, therefore this kind of convolution can computationally handle a large number of output channels compared to ordconv. The number of applied 1D filters per input channel is not strictly bound to 1. Variants of sepconv use a depth multiplier parameter to specify the number of applied 1D filters per input channel however this in turn again increases the required computation power.

## 3.3 Deep Residual Convolutional Neural Networks

Traditional deep CNNs perform quite well in image classification tasks. The number of parameters and therefore the complexity of the network function increases exponentially with increasing depth. Hypothetically an n-layer network should be able to outperform an n-1-layer network regarding the training error or at least be equally good. This comes due to the fact that the complexity of the representing function from the n-layer network is higher than the one from the n-1-layer network. For some intuition, let us take the trained n-1-layer network and add a layer that simply maps the identity function, i.e., the input equals the output. The n-layer network is naturally equally good as the n-1-layer network. As a matter of fact, increasing the depth of a CNN really does show a certain trend of a decreasing training error. However, empirical data has shown that there is a certain network- and task-specific tipping point where the training error starts to increase again. For example, see figure 3.1, where the training and

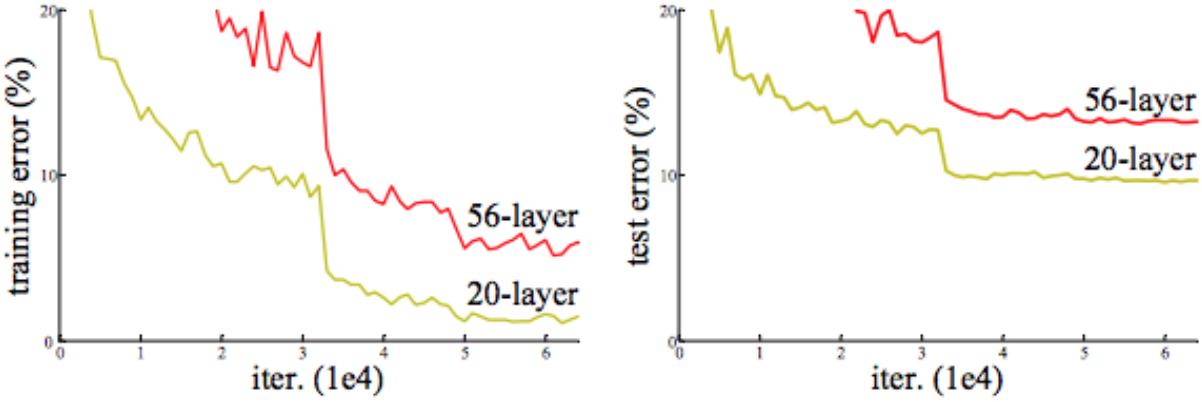


Figure 3.1: Training error (left) and test error (right) on the CIFAR-10 dataset with 20-layer and 56-layer CNNs from this [9] paper.

thus testing error of the shallower network is lower. This phenomenon indicates that not all networks are similarly easy to optimize [9], i.e., that the identity function isn't that easily learned by a layer.

Residual CNNs offer mitigation of this problem where the identity mapping is attempted to be passed along through the whole network. A residual CNN is a CNN with shortcut connections. Shortcut connections are connections that lie between two non-adjacent network layers. The output of a specific preceding layer is usually added elementwise to the input of a specific subsequent layer depending on the network design. That means a fraction of the input values for the next layer has in each case skipped the function transformation of the last or last few layers depending on the architecture see figure 3.2. A comparison example of the training error of a CNN and a residual CNN with the ImageNet dataset can be seen in figure 3.3. Notice how the deeper residual CNN outperforms the shallower residual CNN but the shallower plain CNN outperforms the deeper plain CNN.

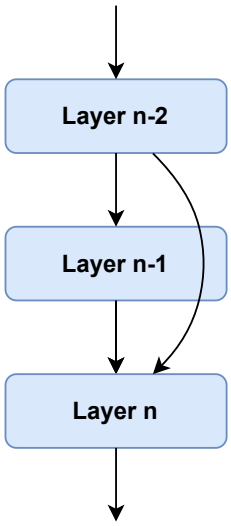


Figure 3.2: Structure of a residual Neural Network: The shortcut brings identity information from layer n-2 to layer n

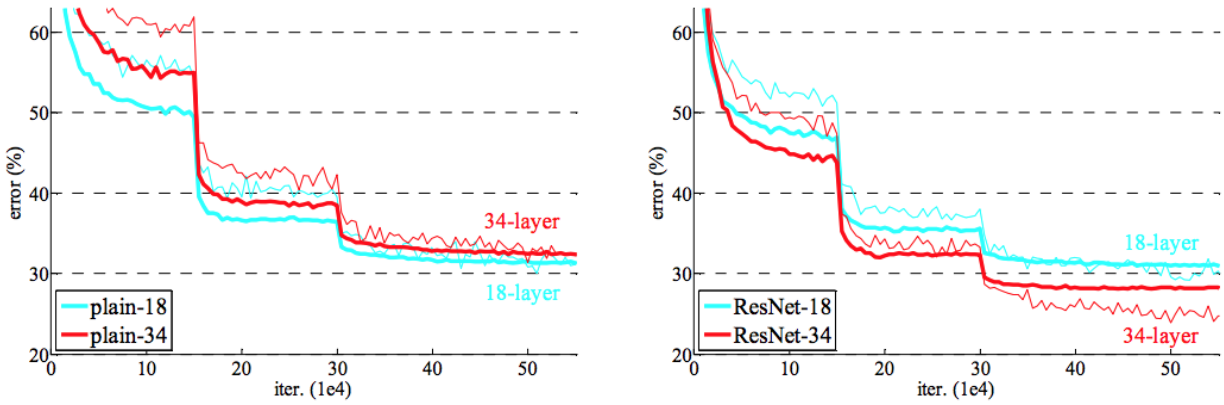


Figure 3.3: Comparison of CNN (left) and residual CNN (right). The thin line represents the training error, and the bold line represents the validation error [9]

.

Chapter 4

# Methodology

This thesis used and built upon David Cleres' code, whose goal was the implementation of Barata et al. 2020's cough detector in Tensorflow version two [12] [8]. This chapter describes the research process and the methods and tools used in this project.

## 4.1 Infrastructure

For the infrastructure, the Euler CPU cluster from ETHZ, along with an NVIDIA GPU cluster from the Management Information Systems Group and the Information Management Group was used. The CPU cluster was mainly used for evaluation, while the GPUs were mainly used for weight optimization.

## 4.2 Data

### 4.2.1 Datasets

For the training, the dataset from Barata et al. 2020 was used in this thesis [8]. Audio from 94 adults with asthma was recorded (mean 43 years; standard deviation 16 years; female: 57%; male: 43%). Out of 704'697 sounds in total, 30'304 were identified as coughs. 650ms cough and noise samples were isolated, centered and labeled. The noise samples were split into five different datasets (A-E) to be used for the different classifiers in the ensemble classifier. All the cough samples remained in one dataset. The five different classifiers that were trained on the different datasets will be referred to as classifier A through to classifier E accordingly throughout this report.

For the examination of the postprocessing stage, a covid-19 dataset was used. It was collected from 10 hospitalized adult covid-19 patients. It contains a total of 3888 cough sounds and was recorded over 24 hours.

### 4.2.2 Datasplits

Using cross-validation in the deep learning task would have been too much of a computational burden and therefore the asthma dataset was split into disjoint sets of training-, validation- and test data. Also no two datasets contained data from the same patient at the same time. The training data was used to optimize the network weights, and the validation data was used for model tuning and selection. The testset was used as a generalizable reference on unseen data. The Validation data was never included as training data.

For the postprocessing regression part, no test data was split off due to the small amount of data. In this task simply leave-one-group-out cross-validation (LOGO-CV) with a group being a patient was used such that there could still be a certain amount of generalizability with the disjoint cross-validation splits.

### 4.2.3 Data Preprocessing

The audio samples of 650ms first were loaded with a sampling rate of 22050 and then min-max normalized such that they only contain values between -1 and 1. After that the librosa library was used to compute the Mel spectrogram with 80 bands, a hop length of 112, an FFT window of 2048 datapoints and as for the exponent of the magnitude the simple magnitude with power one (energy) was used, it was not squared [13]. As there was quite a lot of data, the TFRecords data structure was used to store the processed samples for more efficient memory management.

## 4.3 Model Architecture

### 4.3.1 Optimizer & Loss Function

The Adam optimizer with default parameters and an exponential decay schedule gave the best results regarding the optimizer [14] [15]. Using $10^{-3}$ as the initial learning rate gave rather stable results without any major downward fluctuations over several runs for the architecture without residual connections, therefore this value was not extensively optimized any further. Also, no exhaustive search for the best optimizer was conducted. Adding gradient noise, i.e., using a noisy Adam, was also assessed but seemed to give inferior results [16] in the longitudinal testing. Also, using the Adabelief [17] optimizer was attempted in hope of faster convergence, however the predictive performance seemed to be negatively affected thereby, also no convergence speedup could be observed. This optimizer was however only assessed on the architecture without residual connections. As for the batch size the value 128 also seemed more stable compared to higher or lower

values in the architecture without residual connections and was used throughout this project.

### 4.3.2 Network Structure

The network structure was strongly inspired by Barata et al. 2020's Cough detector with residual connections as key difference in most Experiments. Early layers in the architectures tend to use convolution with only horizontal filters, then transitioning to square filters in deeper layers. This was chosen as primary features were assumed to be a characteristic time-dependent change in amplitude over the different frequencies. In almost all the experiments, mainly horizontal pooling was used instead of two-dimensional pooling between layers as this seemed to have a higher predictive performance.

### 4.3.3 Ensemble Learning

To improve predictive performance an ensemble classifier with five classifiers was trained using the five different noise datasets. As for the cough data, for every classifier, the same cough dataset was used. The aggregation was done via computing the mean of all five cough probabilities. Note that the model architecture tuning and selection was mainly done only on the single classifier that used noise dataset A. The alternative, i.e., the ensemble comparison, would have simply required too much computing power to be able to compare as equally many architectures.

## 4.4 Non-architectural Approach

### 4.4.1 Empirically Derived Postprocessing Rules

Having a strong sample classifier is definitely a major and important part when building a cough detector or rather a cough counter. In deployment however a cough detector's input, unfortunately, does not consist of nicely isolated and centered cough and noise samples. It rather receives long audio recordings that have to be preprocessed first. As the exact location of a cough in the audio recording is unknown to the cough detector, one possible approach is cutting out strongly overlapping windows. That way, an actual cough will be approximately centered with a high probability in at least one window. These windows then would be fed into the classifier and a vector of probabilities would be returned. This approach however additionally requires some postprocessing as the overlapping windows could be detecting a corresponding cough multiple times. The postprocessing stage's task would be extracting the correct number of coughs out of a probability vector. That said, no matter how good the sample classifier is,

if the postprocessing is *not* reliable, so will be the cough count. Barata et al. 2020's empirically derived pre- and postprocessing was used for that matter. The postprocessing rules depend on two thresholds and go as follows: (1) Only two consecutive probabilities above the first threshold are considered to be coughs. (2) Single probabilities are only considered if the mean of the probability in question and the following probability is above the second threshold. (3) Whenever more than eight consecutive probabilities above the first threshold occur, they are considered as two coughs. These two thresholds had to be optimized for each longitudinally tested model individually. This was done first using a slight grid search to get a better sense of the optima, followed by some sort of hill climbing.

This process of extracting the correct number of coughs from the probability vectors has a direct and rather large impact on the labeling correctness of the cough detector. Thus improving this particular process might have a lot of potential to getting closer to this thesis' research goal.

### 4.4.2 Machine Learning Postprocessing

To achieve an improvement of the postprocessing stage, a machine learning approach was targeted. For this task the covid-19 dataset was used as its data format was better fitting for future projects. To create the features and labels, the ensemble classifier of experiment 9 was used to create probability vectors. The differently sized audio files were split up to 6.5-second files and padded with zeros whenever their length was not a multiple of 6.5 seconds. The vectors were also created using the same overlapping window method as mentioned before. The probability vectors were labeled using the cough-timestamps in the textfiles from the annotator's labels. Whenever a cough was located on an edge of a multiple of 6.5 seconds, the cough count was added to the label of the probability vector containing the higher cough share. Only the audio with file-timestamps between 23:00 and 6:59 was used as this application is specifically made for nocturnal monitoring. This resulted in a dataset with 750 remaining cough sounds out of 3888. Furthermore patient four was omitted due to unclean samples i.e. too much background noise etc. Only Patients 1, 2, 3, 5, 6, 7, 8, 9 and 10 were used for this experiment. This resulted in a dataset with 594 remaining cough sounds out of 3888. Only 8 out of those 594 coughs (1,34%) were located on an edge when splitting the data into 6.5-second files. A support vector machine regression model was optimized on this dataset and assessed using LOGO-CV with a group being a patient. The model parameters are specified in subsection 4.6.4. As a reference for improvement, the postprocessing rules were also optimized and evaluated using LOGO-CV on the same data splits. One downside of using this dataset instead of the asthma dataset is that the resulting model can not be included in this project due to the different format and thus not directly be compared with Barata et al. 2020.

## 4.5 Evaluation

For the majority of the experiments only one weight optimization run was conducted and its evaluation was done on the validation set. This section describes in more detail how the evaluation was carried out.

### 4.5.1 Matthews Correlation Coefficient

As evaluation metric, the Matthews correlation coefficient (MCC), a special case of the in statistics known phi coefficient, was used as it is generally regarded as a balanced measure even if the data distribution possesses a strong imbalance [18]. The formula can be seen in equation 4.1, where TP, TN, FP, FN denote true positives, true negatives, false positives and false negatives.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{4.1}$$

### 4.5.2 Thresholding

Thresholding is said to be most effective on classification tasks with highly unbalanced data. As this is the case in our classification task thresholding was used after the training to get a better sense of the model's predictive potential.

### 4.5.3 Validation

Like mentioned, in addition to the validation during the training a thresholded validation was performed after the training to evaluate the single classifiers more thoroughly. This also happened on the validation set, the same one that was also used for model tuning and selection. These datasets still consist of *isolated* cough and noise samples. The datasets' names are "test_cough_final_4" and "test_other_final_4", which can be misleading as they were used as validation data. The best threshold was grid searched among a value between 0 and 1 in distances of 0.005. The ensemble classifiers were likewise evaluated using the same method and the same validation datasets as for the single classifiers.

### 4.5.4 Testing

The most representative evaluation, i.e., the testing, was done on a separate dataset of audio recordings from patients where the cough and noise sounds were *not* specifically isolated. A window with 650ms length was slid over the long audio files with 90% overlap between two samples and in that way generated the data to be predicted or rather tested. After predicting all the probabilities of the testing

samples per night, a few empirically derived postprocessing rules from Barata et al. 2020 were applied to get an exact predicted cough time frame. Then the predicted cough time frames were matched with the labeled cough time frames for every night individually, followed by the computation of their mean difference and its corresponding standard deviation. The test dataset was composed of the following patients: S2, S40, S6, S32, Z29, Z7, Z18, Z44, S27, Z36, Z3, S19, Z24, S45, S46.

## 4.6 Experiments

### 4.6.1 Non-Residual Architectures

#### Experiment 1 (Cleres)

To start, ***Cleres' architecture*** was trained and evaluated as a reference. This architecture can be seen in figure 4.1. A dropout of 0.5 was used before the dense layer. This architecture was attempted to be improved without using residual connections first to get a better sense of what could be achieved in that simpler way.

#### Experiment 2 (Additional Dense)

Adding dense layers to architecture from experiment 1 was assessed, which however did rather lower the MCC. The run with the highest MCC of said architecture was achieved by ***adding three dense 256 layers*** and a dense 2 layer after the global max-pooling.

#### Experiment 3 (Additional Conv)

Adding additional convolutional layers to the architecture of experiment 1 was assessed, which did not improve the MCC either. To assess this, simply ***two additional convolution layers*** with relu like the one from layer three were inserted before the max-pooling of layer three.

#### Experiment 4 (Half Channels)

To see if the computational burden of this model could be reduced without a big decrease in predictive performance, the ***channel size*** of all layers except for the first layer in the architecture from experiment 3 was ***halved***. This achieved similar results as the initial architecture in a few runs. This means the predictive performance was not improved but a similar result was achieved with less of a computational burden.

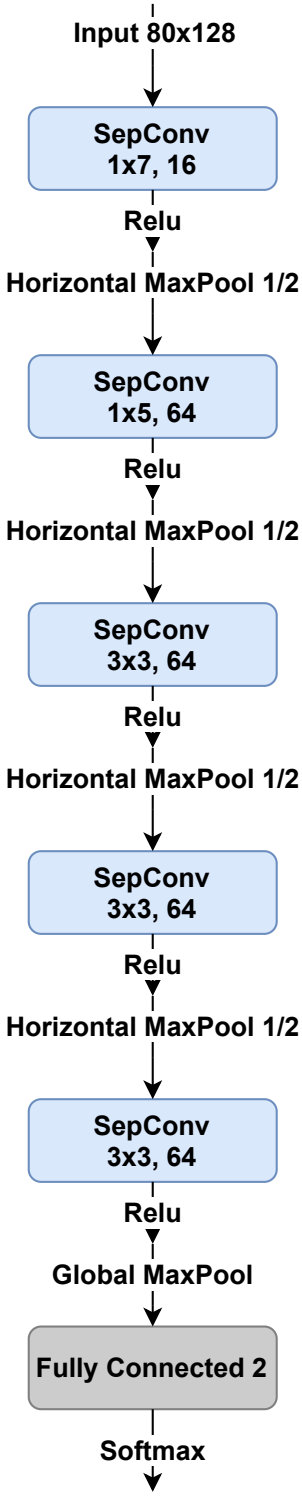


Figure 4.1: Cleres' CNN architecture without residual connections as reference

**Experiment 5 (Flattening)**

The architecture from experiment 4 was also assessed without global max pooling in the end but instead with ***flattening***. Also, restore_best_weights was set to True. This one was the winner among the non-residual architectures.

### 4.6.2 Residual Architectures

**Experiment 6 (L2 Shortcuts)**

For one of the initial ***residual*** architectures, the architecture ***from experiment 1*** was used as a base and simply non-regularized shortcut connections were ***added*** around every convolution layer. Its predictive performance was about equal to the non-residual architecture. Then ***1×1 convolutions*** were used to ***l2 kernel regularize*** the ***shortcuts*** with $\lambda = 5e{-}4$, which improved the thresholded MCC. A run with a higher MCC was one where the first shortcut was simply added without any convolution nor regularization in contrast to the other 4 shortcuts which were all l2 kernel regularized with $\lambda = 5e{-}4$. Its structure can be seen in figure 4.2.

**Experiment 7 (Flattening Residual)**

The next run that achieved a higher predictive performance than the architecture ***of experiment 6*** was when the global max-pooling layer at the end was replaced by a ***flattening*** layer. This run was done with an early-stop patience of 20 and restore_best_weights False.

**Experiment 8 (Average Pool)**

Replacing max-pooling with ***average pooling everywhere*** in the architecture of experiment 7 led to one of the several architectures with the highest MCC. This run was done with an early-stop patience of 10 and restore_best_weights True.

**Experiment 9 (Increased Channels)**

The architecture that felt most stable regarding the predictive performance was achieved by ***increasing the channel size*** and adding ***vertical average pooling twice before*** the ***flattening*** layer. Like in most other experiments, it uses depthwise separable convolution. Also, like in most other experiments, each convolution Layer is skipped with a shortcut connection which performs a 1×1 depthwise separable convolution except for the first layer where the output is

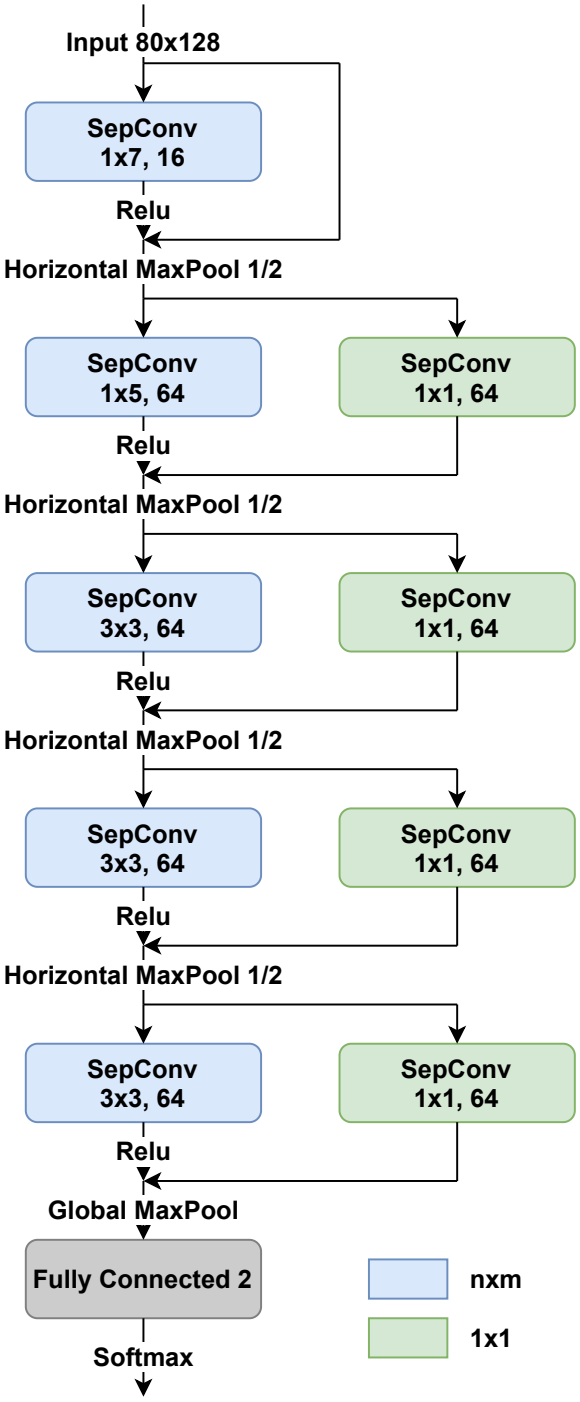


Figure 4.2: First residual CNN architecture that achieved a higher MCC than the non residual architectures. Merging arrows depict the summation of the tensors.

simply duplicated 16 times and added to the input without convolution. All the 1×1 convolutions were l2 kernel regularized with $\lambda = 5e{-}4$. Before the dense layer, a dropout of 0.5 was used. The early-stop parameters used for this experiment were restore_best_weights=True and patience=10. The exact network structure can be seen in figure 4.3.

### 4.6.3 Experiment 10 (Depth Multiplier)

A good run with a very similar architecture was achieved when taking the exact architecture from experiment 9 and changing the ***depth multiplier*** parameter from one to ***two***. And setting the early-stop parameter ***restore_best_weights*** to ***False*** and the early-stop ***patience*** to ***25***. This architecture was the one with the highest testing results from the *very* few architectures that were evaluated on the testset.

#### Experiment 11 (Three 1×3)

This paper [19] suggested replacing early large filters with several smaller filters so the first layer of experiment 9, i.e., the ***1×7*** layer, was ***replaced*** with ***three*** smaller ***1×3*** layers. Also, the two ***vertical pool*** layers were ***removed***. Its exact structure can be seen in figure 4.4.

#### Experiment 12 (Greater Stride)

Also, an architecture inspired by a paper [20] was assessed. This one does not only rely on pooling but also on a ***convolution stride greater than one*** to reduce the resolution. The corresponding architecture can be seen in figure 4.5.

#### Experiment 13 (No Pool)

Another architecture was assessed, where the only means of reducing the resolution was a convolution stride greater than one, meaning ***no pooling*** at all except for global max-pooling at the end. This architecture was also inspired by this [20] paper. Its structure can be seen in figure 4.6.

#### Further Experiments

Many architecture variations were assessed, but the above-named architectures and changes had the most impact on the predictive performance. What could be observed was that depending on the architecture, passing a True or False to the early-stop parameter restore_best_weights made quite a big difference of up to 0.02 MCC. Using mixed precision was also tried originally to have a

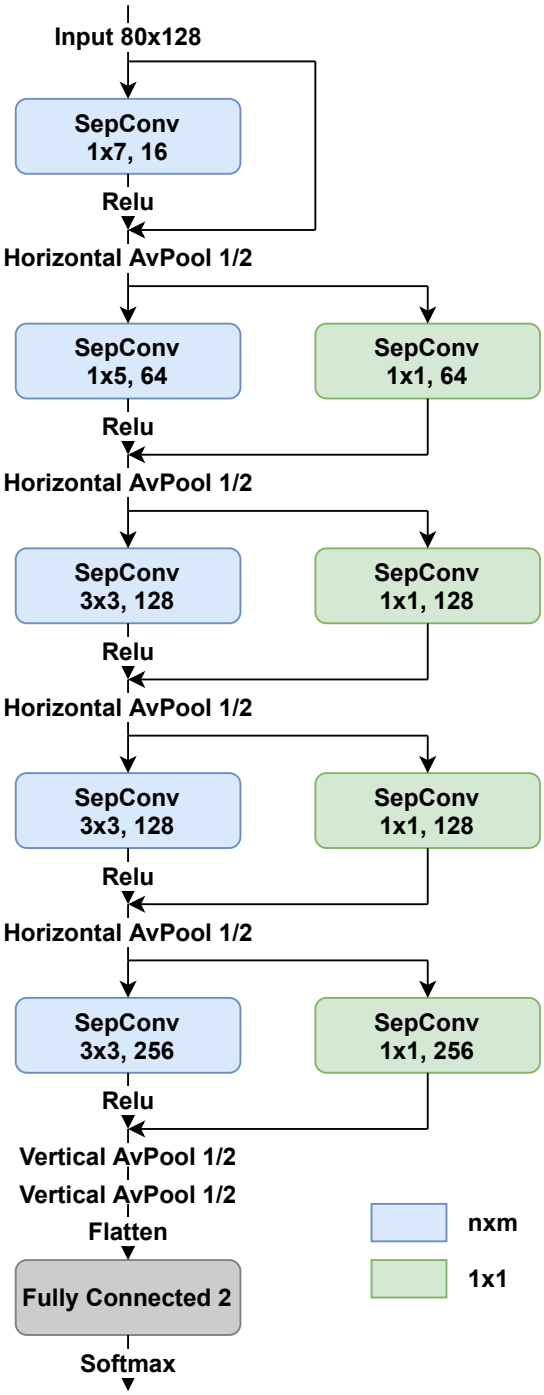


Figure 4.3: Residual CNN structure from experiment 9.

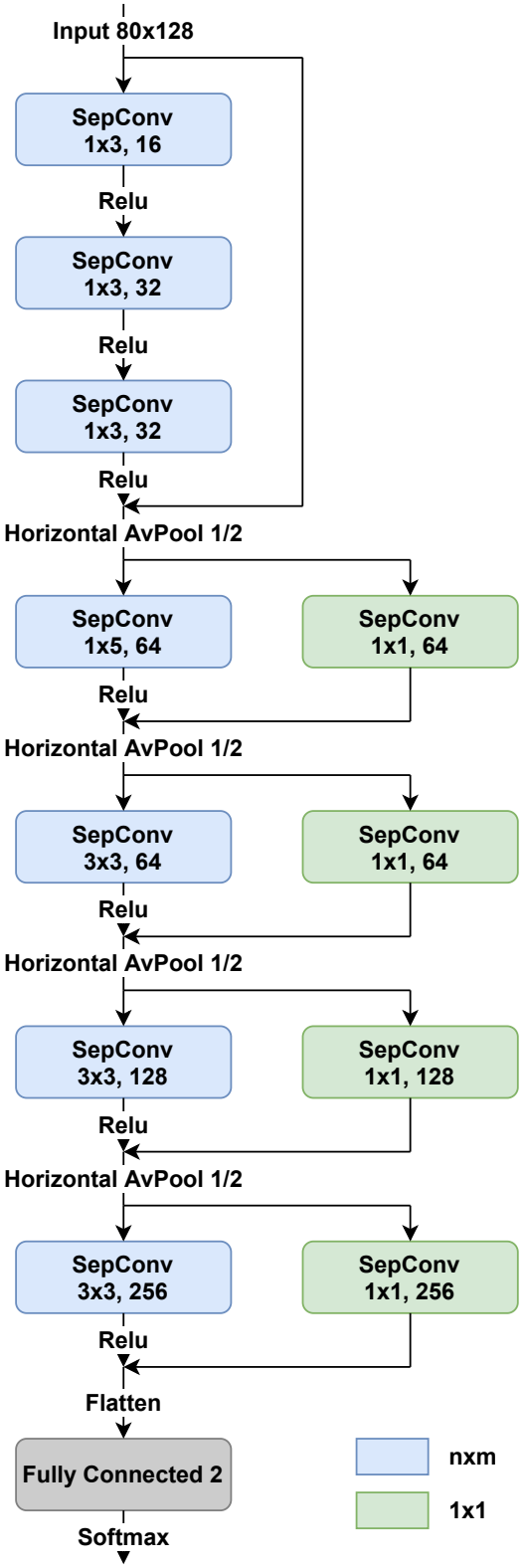


Figure 4.4: Residual CNN structure with three 1×3 convolutions.

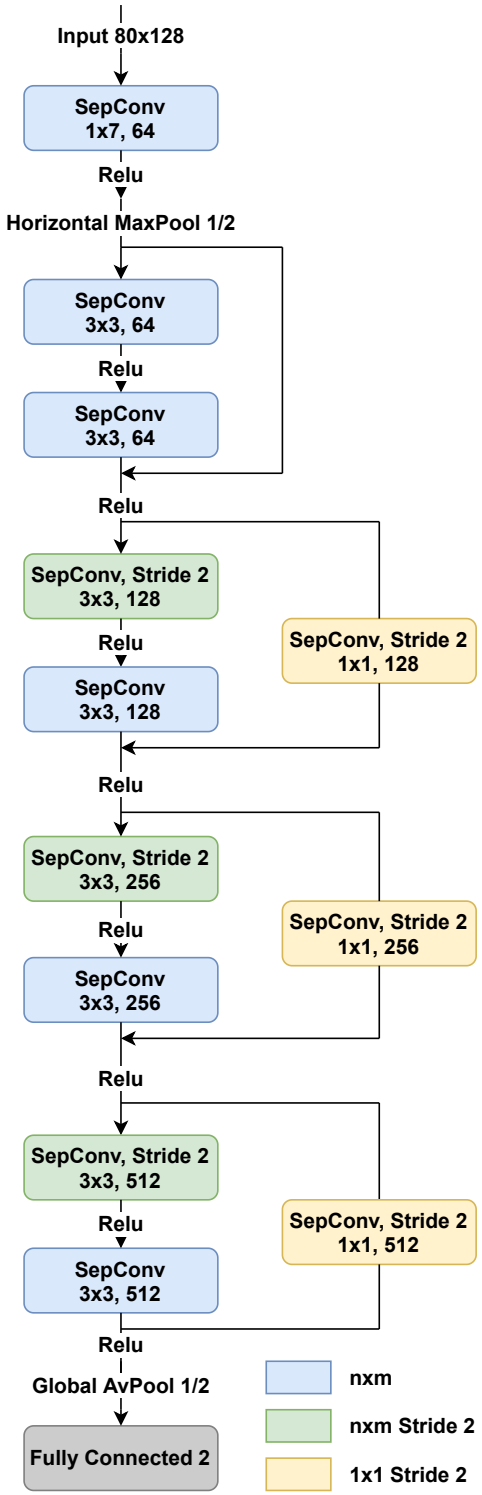


Figure 4.5: Resnet inspired by this paper [20] with pooling only at the very beginning.

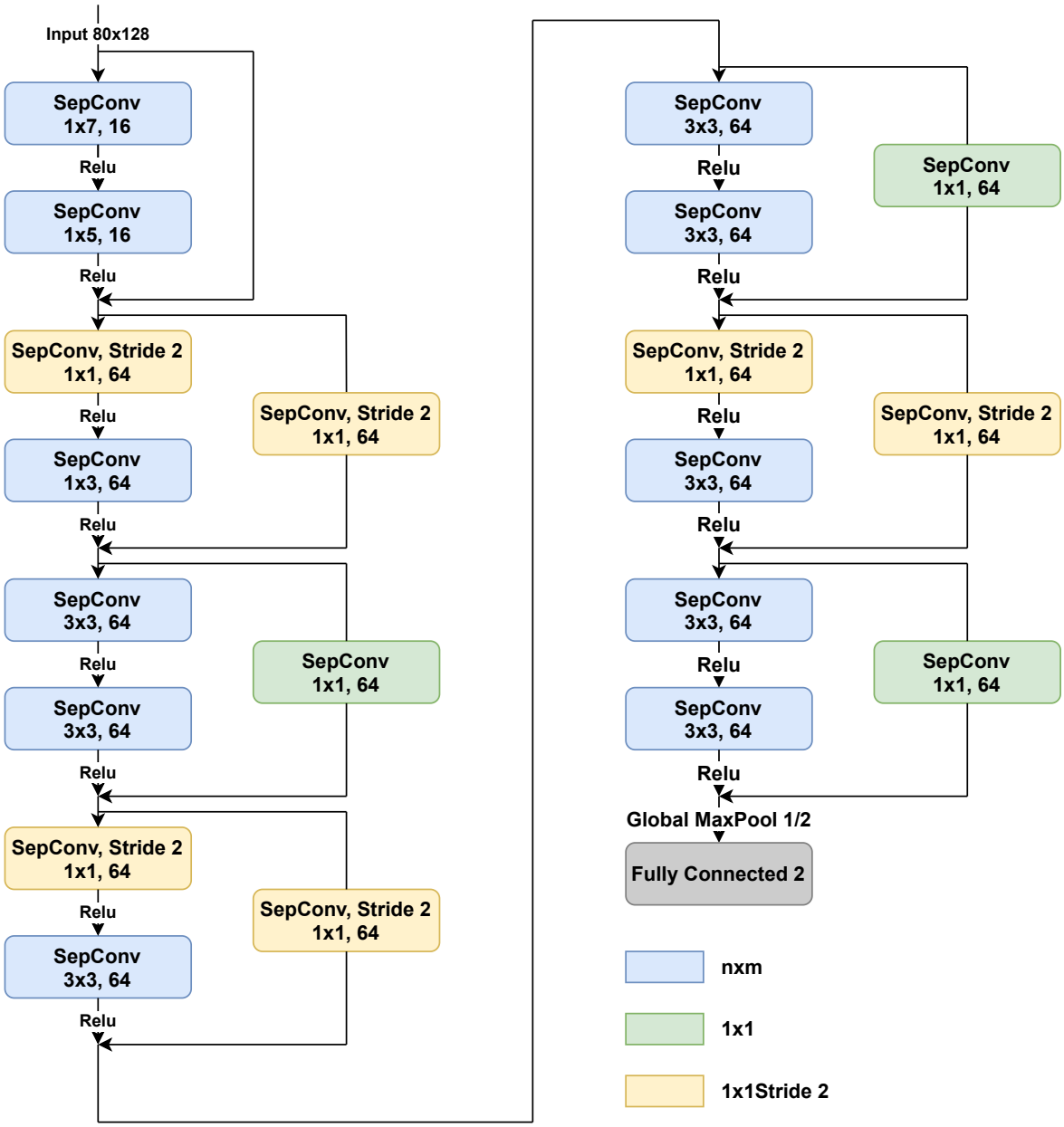


Figure 4.6: Resnet also inspired by this paper [20] without pooling except for the global max-pooling at the end.

faster weight optimization however there was no noticeable speedup and the predictive performance was even negatively affected by up to 0.02 MCC. It has to be said it can not be guaranteed that mixed-precision was used correctly as the application interface was very complicated and seemed to throw one exception after another. An attention module for Keras was also assessed [21]. CBAM, squeeze & excitation, channel attention and spatial attention was each used after each layer. This turned out to slightly worsen the predictive performance, so it was removed, however this approach was not optimized at all due to limited time. Also label smoothing with different hyperparameters was assessed, which also only lowered the predictive performance. As depthwise separable convolution tends to have a lower predictive performance using normal convolutions instead of separable ones was also assessed, which surprisingly also gave worse results. Inserting spatial dropout in early layers was also assessed but did not improve the predictive performance.

### 4.6.4 ML Postprocessing Experiment

For this regression task, a support vector machine regressor was used with an RBF kernel using gamma: 0.41 and a regularization parameter c: 4.3. The scikit-learn framework was used for the regression model as well as for the cross-validation evaluation.

Chapter 5

# Results

## 5.1 Validation Results of Different Architectures

### 5.1.1 Non-Residual Architectures

| Experiment | MCC | Recall | Precision |
|---|---|---|---|
| **1 Cleres** | 0.86[1] | 0.86[1] | 0.86[1] |
| **2 Additional Dense** | 0.86 | 0.86 | 0.87 |
| **3 Additional Conv** | 0.86 | 0.88 | 0.85 |
| **4 Half Channels** | 0.87 | 0.87 | 0.87 |
| **5 Flattening** | 0.88 | 0.87 | 0.90 |

Table 5.1: Results of Experiments with non-residual architectures. The values correspond to the one run of the same architecture with the highest thresholded validation MCC.

### 5.1.2 Residual Architectures

## 5.2 Validation Result of the best Ensemble Classifier

A few promising architectures were also trained on the other four datasets and evaluated as an ensemble classifier. The thresholded MCC of the architecture from experiment 9 was 0.925 with a recall of 0.926 and a precision of 0.926. Assessing the chosen architecture with added gradient noise with noise_eta=1 resulted in an even higher validation MCC of 0.928, however the longitudinal test results were lower with the added gradient noise. Also, instead of mean value aggregation, inverse entropy and majority vote aggregation was assessed, but they both turned out to have a lower thresholded ensemble validation MCC.

[1]These values do not correspond to the run with the highest MCC of this architecture, they approximately represent the average value over several runs.

[2]The average MCC over several runs of this architecture was only 0.9 approximately.

| Experiment | MCC | Precision | Recall |
|---|---|---|---|
| **6 L2 Shortcuts** | 0.89 | 0.88 | 0.89 |
| **7 Flattening Residual** | 0.90 | 0.88 | 0.92 |
| **8 Average Pool** | 0.91 | 0.89 | 0.92 |
| **9 Increased Channels** | 0.91[2] | 0.91 | 0.91 |
| **10 Depth multiplier** | 0.91 | 0.90 | 0.92 |
| **11 Three 1×3** | 0.91 | 0.91 | 0.92 |
| **12 Greater Stride** | 0.89 | 0.90 | 0.88 |
| **13 No Pool** | 0.90 | 0.88 | 0.92 |

Table 5.2: Results of Experiments with residual architectures. The values correspond to the one run of the same architecture with the highest thresholded validation MCC.

The architecture of experiment 10 gave a thresholded MCC of 0.911 with a recall of 0.910 and a precision of 0.914 which was lower but gave better test results.

## 5.3 Longitudinal Testing Results

The classifier from experiment 9 was assessed on the longitudinal test set. Its Bland-Altman plot and histogram with the two thresholds 0.76 and 0.61 can be seen in figures 5.1 and 5.2. Its Bland-Altman plot and histogram with the two thresholds 0.71 and 0.59 can be seen in figure 5.3 and 5.4.

Also the classifier from experiment 10 was assessed on the longitudinal test set. Its Bland-Altman plot and histogram with the two thresholds 0.69 and 0.5658 can be seen in figures 5.5 and 5.6. Its Bland-Altman plot and histogram with the two thresholds 0.69 and 0.5727 can be seen in figure 5.7 and 5.8. Its Bland-Altman plot and histogram with the two thresholds 0.71 and 0.5751 can be seen in figure 5.9 and 5.10. Its Bland-Altman plot and histogram with the two thresholds 0.7 and 0.623 can be seen in figure 5.11 and 5.12.

## 5.4 ML Postprocessing Results

The cross-validation MCC of the regression model was 0.744. The cross-validation MCC of the postprocessing rules was 0.728. The MCC of the postprocessing rules that were optimized on all 9 patients without cross-validation was 0.778 with a first threshold of 0.587 and a second threshold of 0.473.

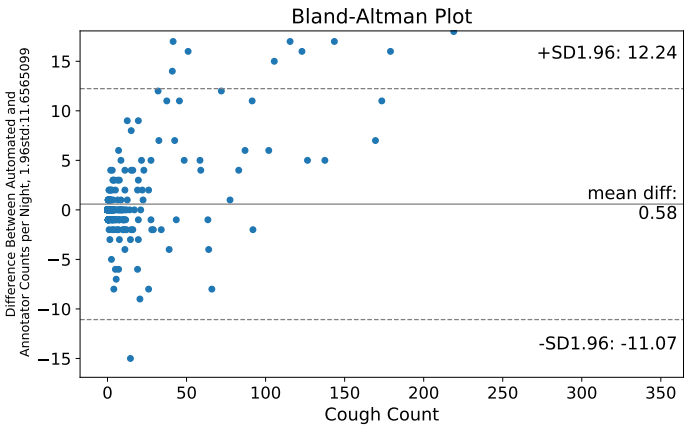


Figure 5.1: Bland-Altman plot of architecture from experiment 9 with thresholds 0.76 and 0.61

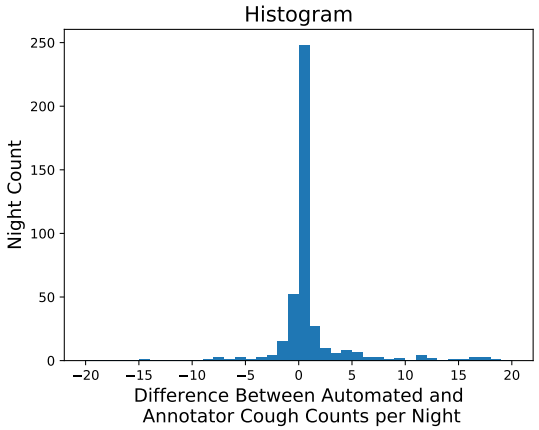


Figure 5.2: histogram of architecture from experiment 9 with thresholds 0.76 and 0.61

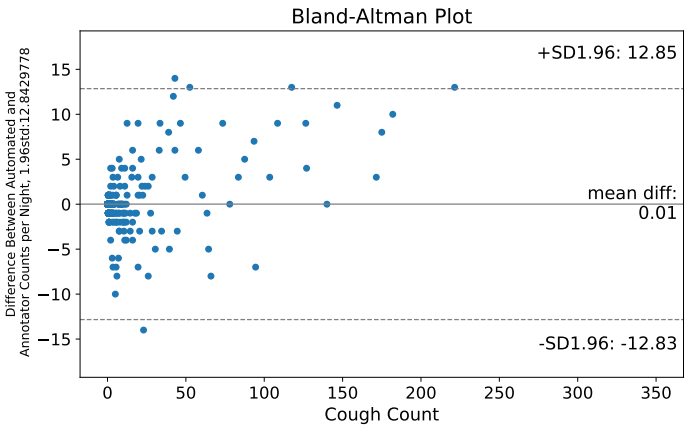


Figure 5.3: Bland-Altman plot of architecture from experiment 9 with thresholds 0.71 and 0.59

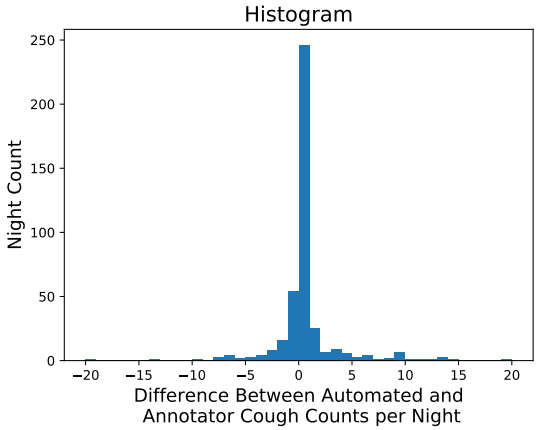


Figure 5.4: histogram of architecture from experiment 9 with thresholds 0.71 and 0.59

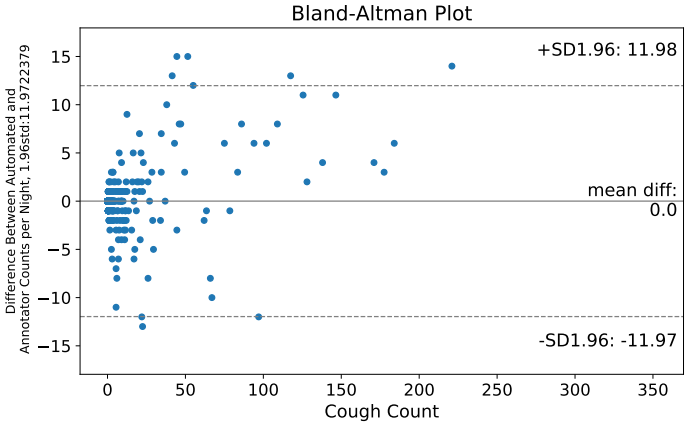


Figure 5.5: Bland-Altman plot of architecture from experiment 10 with thresholds 0.69 and 0.5658

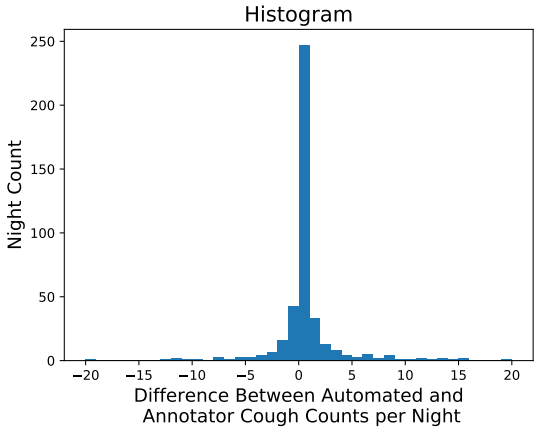


Figure 5.6: Histogram of architecture from experiment 10 with thresholds 0.69 and 0.5658

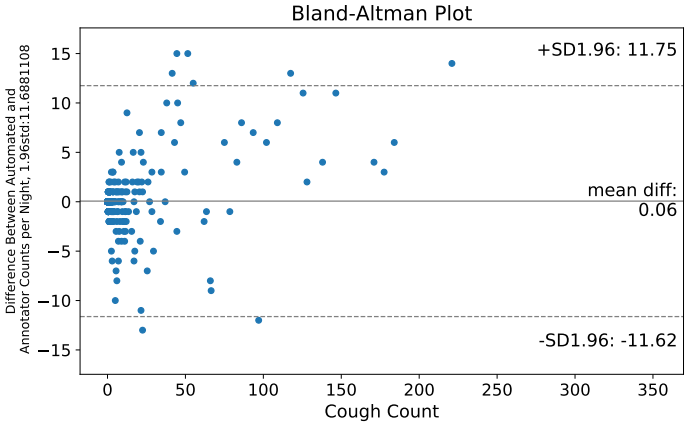


Figure 5.7: Bland-Altman plot of architecture from experiment 10 with thresholds 0.69 and 0.5727

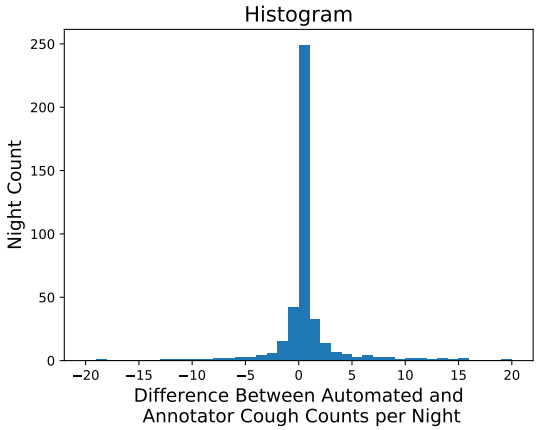


Figure 5.8: Histogram of architecture from experiment 10 with thresholds 0.69 and 0.5727

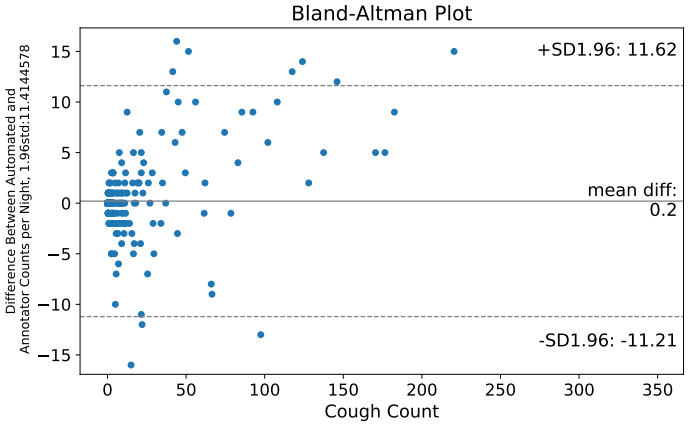


Figure 5.9: Bland-Altman plot of architecture from experiment 10 with thresholds 0.71 and 0.5751

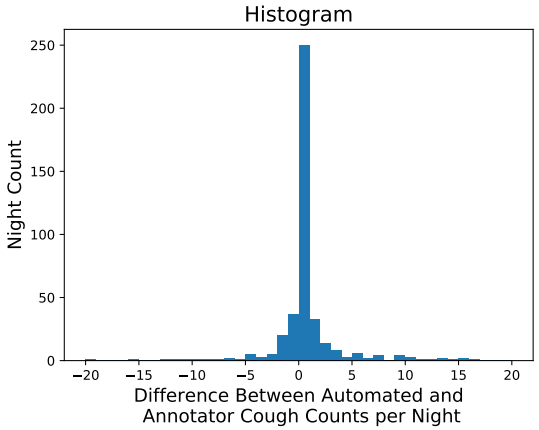


Figure 5.10: Histogram of architecture from experiment 10 with thresholds 0.71 and 0.5751

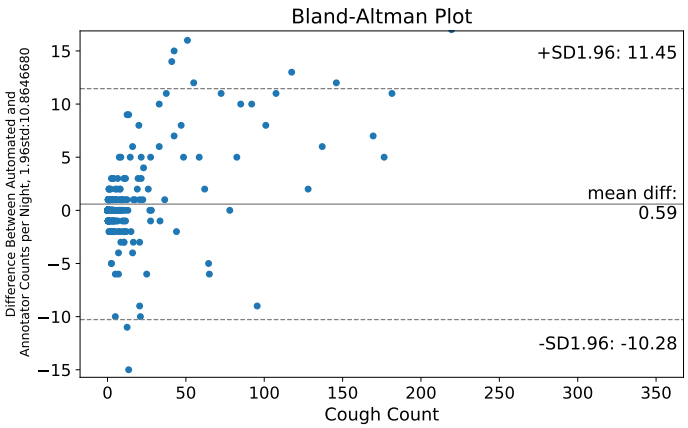


Figure 5.11: Bland-Altman plot of architecture from experiment 10 with thresholds 0.70 and 0.6230

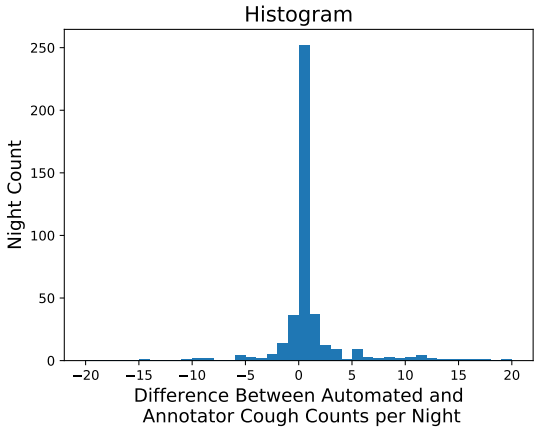


Figure 5.12: Histogram of architecture from experiment 10 with thresholds 0.70 and 0.6230

Chapter 6

# Discussion

## 6.1 Principal Findings & Practical Implications

With regard to the research goal mentioned in the introduction, the goal of achieving better Results than Barata et al. 2020's cough detector [8] with the newer TensorFlow framework version two possibly using deep residual convolutional neural networks and other non-architectural approaches was accomplished. The residual architecture was built, trained and assessed and did perform better predictively than our reference architecture without residual connections. A big jump in predictive performance was noticed when replacing global max-pooling by flattening. The same goes for replacing max-pooling between layers with average pooling. Barata et al. 2020's count difference per day between automated and human-annotated coughs was a mean of –0.1 coughs with a 95% confidence interval of –12.11 and 11.91 coughs. And our detector achieved a mean of 0 coughs with a 95% confidence interval of –11.97 and 11.98 coughs or with other thresholds a mean of 0.06 coughs with a 95% confidence interval of -11.62 and 11.75 coughs. The mean difference and the confidence interval are both lower compared to Barata et al. 2020's results. The best confidence interval was achieved with the thresholds of 0.7 and 0.623, but with these thresholds the mean difference was higher compared to Barata et al. 2020. The computational burden of our model is much higher compared to Barata et al. 2020 where a single classifier needed 10.74 million floating point operations. Our model from experiment 9 needs 286 million and our model from experiment 10 needs 431 million floating point operations for a prediction with a single classifier. This might not be such a big problem as the computing power of smartphones keeps increasing. The number of trainable parameters of model 9 is 203'145 and 264'496 for model 10. This could also be of importance as smartphone applications usually have a size limit. Referring back to the initially set research goal, unfortunately, it can not directly be measured to what extent the improvement of the postprocessing stage would also improve the cough detector. They are a step in the same direction however. The cross-validated score on the nine patients was improved by 0.016 MCC (+2.1%) compared to the optimized postprocessing rules. This might not be a major improvement yet, but it implicates that it can yet be improved with-

out major effort. This would directly affect or rather improve the deployed cough detector even more.

## 6.2 Limitations & Potential Future Improvement

The model architecture optimization problem of the cough classifier has a very large number of possible architectural-, optimization method- and hyperparameter choices. Extensive optimization of every single aspect was not conducted as it would have required a much larger amount of time and computing power than available.

Another point to mention is the majority of the experiments were not evaluated as ensemble classifier but merely as classifier A. The evaluation of every experiment as ensemble might have been more representative as the classification used in the detector pipeline is the ensemble and not the single classifier. This however would also have required more time and computing power.

Additionally, the CNN's initialization and weight optimization is a process that depends on highly non-deterministic algorithms, therefore to have representative evaluation results, one would have to do several runs over every experiment and then take the average thereof as the actual result. This would have contributed even more to the mentioned limited time and computation issue.

In the majority of experiments, only one run was conducted and only classifier A was trained and evaluated, therefore the results should be interpreted with caution.

Due to time limitations only the data from the asthma study was used in this project even though additional data from the covid 19 study was available. In future work, the datasets could be combined to improve the predictive performance of the model even more. Also the Validation data could be included into the training data before testing, which was not done. This might improve the testing results due to the larger dataset. Regarding the data, also *no* outlier detection and removal was conducted which usually has a big impact and could improve the detector in future work.

Optimizing a model architecture in a *systematic* way proved to be more challenging than originally expected. For future work, it might be rewarding to plan some time first to think of a good system.

Data augmentation with non-deterministic window offsets instead of centered windows was not assessed in this thesis and might be a potential improvement for future work. As a result the window overlap in the preprocessing might even be reduced which in turn would take some load of the computational burden.

Also, a few approaches or hyperparameters that were not exhaustively enough assessed are the depth multiplier of the separable convolution, early stop condi-

tions (in particular patience and restore_best_weights), the attention module, dropout, spatial dropout, label smoothing and mixed precision. Using the Mel frequency cepstral coefficients instead of the Mel spectrogram might also have some potential and taking the squared magnitude of the Mel spectrogram instead of only the magnitude could also make some difference.

Another way of improving the cough detector could be improving or using the improved postprocessing in future work. This way could be less challenging compared to other approaches as this has not been optimized as much as other approaches yet. Also, the ML postprocessing experiment implicates that it is possible.

A possible approach from a quite different angle would be using Recurrent Neural Networks. This would save the need of postprocessing and its optimization and possibly also of the preprocessing and its optimization.

Chapter 7

# Conclusion

To answer the initially posed research questions, we *were* able to improve neural network-based cough detection using residual neural networks with Barata et al. 2020 as a reference. The extent of which can be seen comparing the longitudinal test results. The detector was improved from a per day mean difference of –0.1 coughs with a 95% confidence interval of –12.11 and 11.91 coughs to a mean difference of 0.06 coughs with a 95% confidence interval of –11.62 and 11.75 coughs or a mean difference of 0 coughs with a 95% confidence interval of −11.97 and 11.98 coughs depending on the chosen thresholds. Furthermore, a postprocessing regressor with a 0.016 higher (+2.1%) LOGO-CV MCC score compared to the previous postprocessing rules implicates that the postprocessing of Barata et al. 2020 can be improved. Regarding the second research question, unfortunately due to the evaluation on two different datasets, it can not directly be measured to what extent machine learning-based postprocessing can improve predicted cough counts. If the different postprocessing approach could have been included in the testing pipeline the above mentioned testing results would have with a high chance even been higher.

# Bibliography


[1] A. Morice, G. Fontana, M. Belvisi, S. Birring, K. Chung, P. Dicpinigaitis, J. Kastelik, L. McGarvey, J. Smith, and M. T. et al, "Ers guidelines on the assessment of cough," *European respiratory journal*, vol. 29, no. 6, pp. 1256–1276, 2007.

[2] J. Smith and A. Woodcock, "Cough and its importance in copd," *International journal of chronic obstructive pulmonary disease*, vol. 1, no. 3, p. 305, 2006.

[3] F. Barata, K. Kipfer, M. Weber, P. Tinschert, E. Fleisch, and T. Kowatsch, "Towards device-agnostic mobile cough detection with convolutional neural networks," in *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, 2019, pp. 1–11.

[4] "Pulmonary edema pennmedicine," https://www.pennmedicine.org/for-patients-and-visitors/patient-information/conditions-treated-a-to-z/pulmonary-edema, accessed: 2021-12-01.

[5] "Heart failure michiganhealth," https://www.uofmhealth.org/health-library/hw44415, accessed: 2021-12-01.

[6] A. Windmon, M. Minakshi, P. Bharti, S. Chellappan, M. Johansson, B. A. Jenkins, and P. R. Athilingam, "Tussiswatch: A smart-phone system to identify cough episodes as early symptoms of chronic obstructive pulmonary disease and congestive heart failure," *IEEE Journal of Biomedical and Health Informatics*, vol. 23, no. 4, pp. 1566–1573, 2019.

[7] S. S. Birring, T. Fleming, S. Matos, A. A. Raj, D. H. Evans, and I. D. Pavord, "The leicester cough monitor: preliminary validation of an automated cough detection system in chronic cough," *European Respiratory Journal*, vol. 31, no. 5, pp. 1013–1018, 2008. [Online]. Available: https://erj.ersjournals.com/content/31/5/1013

[8] F. Barata, P. Tinschert, F. Rassouli, C. Steurer-Stey, E. Fleisch, M. A. Puhan, M. Brutsche, D. Kotz, and T. Kowatsch, "Automatic recognition, segmentation, and sex assignment of nocturnal asthmatic coughs and cough epochs in smartphone audio recordings: Observational field study," *J Med Internet Res*, vol. 22, no. 7, p. e18082, Jul 2020. [Online]. Available: https://doi.org/10.2196/18082

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[10] "Video stft," https://www.youtube.com/watch?v=-Yxj3yfvY-4, accessed: 2021-12-09.

[11] "Video melspectrogram," https://www.youtube.com/watch?v=9GHCiiDLHQ4&t=1170s, accessed: 2021-12-09.

[12] "David cleres' original project on github," https://github.com/ADAMMA-CDHI-ETH-Zurich/CoughDetection, accessed: 2021-12-06.

[13] "Librosa library," https://librosa.org/doc/latest/index.html, accessed: 2021-12-06.

[14] "Adam optimizer from tensorflow," https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam, accessed: 2021-12-06.

[15] "Exponential decay schedule," https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/ExponentialDecay, accessed: 2021-12-06.

[16] "Gradient noise library," https://github.com/cpury/keras_gradient_noise, accessed: 2021-12-06.

[17] "Adabelief optimizer library," https://github.com/juntang-zhuang/Adabelief-Optimizer, accessed: 2021-12-06.

[18] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, p. 6, Jan 2020. [Online]. Available: https://doi.org/10.1186/s12864-019-6413-7

[19] I. Bello, W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T.-Y. Lin, J. Shlens, and B. Zoph, "Revisiting resnets: Improved training and scaling strategies," 2021.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[21] "Attention module keras," https://github.com/kobiso/CBAM-keras/blob/master/models/attention_module.py, accessed: 2021-12-07.