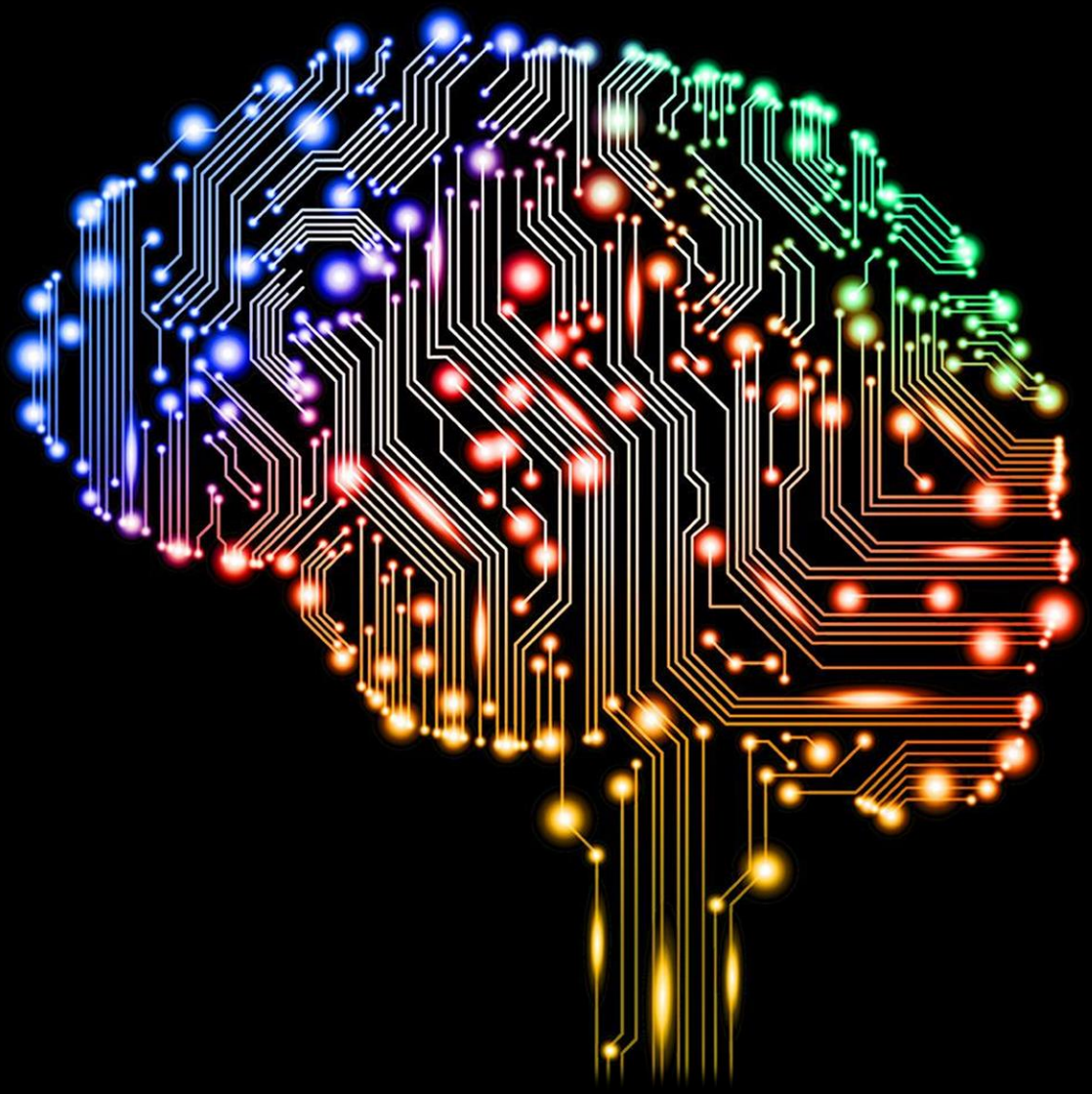
SENNSORS

a voice recognizer

January 21, 2019

Steven van den Wildenberg and Matthijs Ates, 6F

Supervisor: Rachel Baan, Informatica, Atheneum College Hageveld



“Everyone should learn how to create a neural network, because it teaches you how you think.”

~ Matthijs Ates

Index

1. Abstract

Voice recognition is a new and mostly unexplored area of Artificial Intelligence. Voices are unique and very complex information that, if integrated properly, could be used for speaker identification. Current programs seem to be lacking in their accuracy, often approving strangers or disapproving the real calibrator. This paper dives into the recognition of unique voices using the Fourier Transformation, a technique to differentiate voices by frequency. Predicting the speakers has been achieved by using a Deep Neural Network. Consequently, a Smart and Effective Neural Network for Sensing and Ordering Recorded Sounds was created. A program that is able to classify speakers based on a three second audio input with over 99% accuracy for two speakers.

2. Preface

Choosing the topic of Artificial Intelligence was not a hard decision to make. Early 2018, conversations we had created a big fascination for what Artificial Intelligence does now and what it could be able to do in the future. This, in addition to the thought experiment of Roko's basilisk, the idea that a future all powerful AI will punish those who did not contribute to its existence, led to an easy decision of the topic. However, Artificial Intelligence is a huge topic, so making a decision on what exactly to research within the field was not as easy. Weeks of brain storming went by and suggestions like the recognition of handwritten digits or solving a Rubik's cube were in the back of our minds. These ideas just seemed to be quite standard and our excitement seemed to drain away thinking about researching something that had been done hundreds of times before. Was it really research if everything we needed could be copied and pasted directly from the internet?

Suddenly an idea popped into our heads: speaker recognition. We found out it was the less popular sibling of speech recognition. A couple searches on Google confirmed our expectation that there had not been conducted a lot of research on this subject, which was exactly what we wanted. Aside from the fact that a quote from Albert Einstein "If we knew what it was we were doing, it would not be called research, would it?" became very relatable, the idea that we actually did research on Artificial Intelligence made us enthusiastic. In the end, we are superb with the results, especially because something that seems so inaccessible and advanced as Artificial Intelligence turns out to be quite doable for two teenagers in high school.

Furthermore, the research would not have been possible if it was not for the people surrounding us. Special thanks to Rachel for being our supervisor, Sjoerd Offerhaus for directing us towards the Fourier Transformation, Henriet for recording extra test data and our families for the support.

3. Introduction

Often, people with little knowledge of artificial intelligence (AI) get the idea that it will take over the world in the future. We're not going to argue with that, but before it happens, a couple decades will probably go by. In the meantime, we want to help AI evolve into something for a good cause and although this might be a little contribution for the whole industry, there is a chance we set somebody else or ourselves up to create big projects in the future.

As you may have realized, our research topic is artificial intelligence or more specifically machine learning. Machine Learning is a subtopic of AI that focuses on making machines 'learn', but it could basically be seen as mathematically improving the outcome. If we go even more specific our research topic is voice recognition or speaker recognition. These terms sometimes get confused with speech recognition. They mean, respectively, the identification or verification of a voice and the recognition of spoken words. We will use 'speaker recognition' or 'voice recognition' throughout this paper to be clear in what we want to say. Remarkably, speech recognition is already widely used and researched in voice assistants and other fields. This in contrast to speaker recognition, which has not been researched a lot and is also used less in, again, voice assistants.

With personalization among devices and products becoming more important, speaker recognition is a feature that fits perfectly in the picture of optimized user experience. The technology could be perfectly used in voice assistants such as Siri and Alexa. With this research we hope to bring this technology one step closer to the public.

4. Thesis

Creating a scientific paper out of nowhere is not an easy task. Especially in this period of time where the research on Artificial Intelligence is huge. Most research for starters has already been done before like the recognition of handwritten digits. Finding an original topic can be hard but persistence wins. After the general topic of Speaker Identification had been formed, it was not so hard to create a thesis which was in compliance with research in similar fields: **Is it possible to create a deep neural network that can recognize a voice with an accuracy above 90 percent?**

4.1 Partial questions

From this thesis question a few partial questions can be formulated as well.

1. Will the accuracy of the predictions remain the same when a third speaker is added?

2. How does a decreased number of neurons affect the accuracy of the predictions?

4.2 Hypothesis

The hypothesis we have formulated for this question is as follows: **The neural network should be able to achieve an accuracy of 90% in recognizing known speakers**.

5. Artificial Intelligence

In this chapter we will look at artificial intelligence. We will explain what AI is, how it learns and go deeper into the techniques we used for our research. In addition, there is a section on the programming language Python, where we explain the basics of how the programming language works as well as the reasons behind the usage of this specific language for our research.

5.1 Artificial Intelligence in general

An app that can compute the fastest route to your destination, a smartphone that can listen and interpret to what you say, a program that can predict what series you like most, software that can correct spelling mistakes, a site that translates any sentence from any language, to any language. The last century computers have gone from large rudimentary calculators to tiny processors doing billions of computations every second[1]. A smartphone today has more computing power than all of NASA had when they performed the Moon landing in 1969[2]. Computers are becoming better, faster and smarter every year, but what does it mean to have a smart computer?

The first paragraph provides a couple examples of artificial intelligence, or AI in short. We use AI in our day to day lives even without noticing. Every time we use an online translation program, voice assistant or online marketplace, there is AI working in the background.

So, what exactly is Artificial Intelligence? Is the computer becoming self-conscious? How does it work? AI is becoming widely used, the technology companies are just a small part of where it is applicable. Not only the use, but also the jobs related to AI increase every day. According to the Merriam-Webster[3] dictionary, AI is defined as “a branch of computer science dealing with the simulation of intelligent behavior in computers” and “the capability of a machine to imitate intelligent human behavior”. This is a very broad definition of Artificial Intelligence, for the subject of AI itself is very broad in what it can do and how it functions. We will dive deeper into the definition of artificial intelligence in the next section.

In general, AI can be useful for tasks that normally require a human because simply programming an algorithm is hard for tasks related to human senses. Nevertheless, AI can also

[1] Ullah, Z. (2012). Early Computer VS Modern Computer: A Comparitive Study and an Approach to Advance Computer. Retrieved on 4 november 2018, from http://www.academia.edu/29957529/Early_Computer_VS_Modern_Computer_A_Comparitive_Study_and_an_Approach_to_Advance_Computer

[2] Kaku, M. (2011, 15 march). Physics of the Future: How Science Will Shape Human Destiny and Our Daily Lives by the Year 2100. Retrieved on 4 november 2018, from https://www.goodreads.com/work/quotes/13358451-physics-of-the-future-how-science-will-shape-human-destiny-and-our-dail

[3] Merriam-Webster. (n.d.). Definition of Artificial Intelligence. Retrieved on 7 december 2018, from https://www.merriam-webster.com/dictionary/artificial%20intelligence

be quite useful for tasks with a lot of data. The more difficult the task, the more complex the AI. Netflix uses AI to predict what movies you are likely to watch in order to fill your recommended feed. The software behind this uses a lot of data from other users and yourself to compile your personalized feed. It then presents the movies it thinks you will like most or that are closely related to something you most recently watched.

AI uses previously gained knowledge to (accurately) predict the outcome, meaning the AI Netflix uses can, in a way, 'learn' about the person watching. When an AI is able to make decisions based on data it has been given, or that it has gathered itself, it is called machine learning[4].

5.2 Machine Learning

Machine Learning (ML) is not the same as AI. It is specifically about the 'learning part' of AI, the improvement of systems in doing their designated task. Machine Learning is not the same as human learning. Humans can learn new skills and improve the skills they already have, whereas Machine Learning, as of now, can only learn how to do a single task, dependent on the input data it has been given. Machine learning focuses on creating predictions based on data and improving itself with experience. But how does a program do this?

There are many different types of AI, all of which are structured differently and are used to perform different tasks. Consequently, there are as many, if not more, kinds of machine learning. The broadest distinction that can be made between these types is to classify them as either supervised or unsupervised learning.

These two 'classes' of AI learning are distinct in one main part. Supervised learning requires the input to be labeled or classified by a human first, so that the AI knows whether or not its output is correct[5]. Unsupervised learning is its exact opposite. Unsupervised learning algorithms work with data that has not been labeled beforehand. Instead these types of machine learning are programmed to look for patterns in data and work by way of categorizing, or 'clustering' datasets[6].

[4] Fagella, D. (2018, 29 october). What is Machine Learning? Retrieved on 8 november 2018, from https://www.techemergence.com/what-is-machine-learning/

[5] Brownlee, J. (2016, 22 september). Supervised and Unsupervised Machine Learning Algorithms. Retrieved on 8 november 2018, from https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

[6] Maini, V. (2018, 16 august). Machine Learning for Humans, Part 3: Unsupervised Learning. Retrieved on 8 november 2018, from https://medium.com/machine-learning-for-humans/unsupervised-learning-f45587588294

Despite their differences, all machine learning systems work by way of three main components, as stated by the Department of Computer Science and Engineering of the University of Washington[7]. These three components are representation, evaluation and optimization.

In short, representation is the part of AI that takes the input and computes the output. evaluation, or the evaluation function, is the function in the algorithm that discerns the bad outputs from the good outputs. Lastly, optimization is the component that optimizes, or 'improves', the AI in order to get better results the next time[8].

Each of these three components can be filled in by a number of different functions and algorithms. For each task different options can be used, yet not every option is sensible to use in a given situation. For example, a logic program can be very useful for a food distribution system, however, this may not prove very effective in profiling customers at a supermarket, in which case a neural network can be of better use.

Which type of AI and which type of learning mechanism is used is dependent on the task the program has to perform. It is important to define for yourself what the problem is and what type of program can solve it, thereby filling in the three components (representation, evaluation, optimization) in the best way to solve said problem[9].

5.3 Machine learning related to our problem

Speaker recognition is a new and as of yet rather unexplored area of research. Voices are complex and to make a program that is able to recognize and classify voices is therefore no easy task. The data we use as input is large yet shows clear signs of distinction between individuals. Our program has a few requirements to meet, including the abilities to:

- Differentiate between voices
- Improve itself to differentiate more effectively
- Classify new audio samples from known speakers correctly

Consequently this program has to be able to find patterns in the input. This indicates that an unsupervised learning algorithm has to be used. However, the program also needs to know how many classes are possible, and in which class to place new input. Thus, we also must label the input in order to contain the number of classes, meaning that we will use a type of AI able to recognize data structures but has supervised learning algorithms in place to achieve this.

[7] Domingos, P. (2012, 1 october). A Few Useful Things to Know about Machine Learning [Scholarly article]. Retrieved on 8 november 2018, from https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf
[8] Domingos, P. (2012, 1 october). A Few Useful Things to Know about Machine Learning [Scholarly article]. Retrieved on 8 november 2018, from https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf
[9] Domingos, P. (2012, 1 october). A Few Useful Things to Know about Machine Learning [Scholarly article]. Retrieved on 8 november 2018, from https://homes.cs.washington.edu/~pedrod/papers/cacm12

Given our problem we have decided to use a structure called a neural net. This is because it has "a remarkable ability to derive meaning from complicated or imprecise data"[10]. As our data is indeed complicated and, at times, can be rather imprecise (or overlapping) a neural network sounds like the ideal solution to our problem. We have therefore decided to make use of (deep) neural networks to classify voices. How neural networks function will be explained in the next section.

5.4 Neural networks

A neural net is a design of artificial intelligence that is based on the human brain. The human brain is something scientists have been studying since the sixteenth century[11] and research in the exact workings of the mind is still going on. Brains can be viewed as incredibly potent computers. Computer scientists have recognized this as well and have tried to replicate its architecture as a form of AI, creating the first artificial neural networks. But to understand how an artificial neural net functions, we need to know the workings of what it was all derived from.

The human brain functions by way of an incredibly vast network of long cells, called neurons. Neurons consist of two main parts: the cell body, or soma, and the extension, or axon[12]. The exons of these neurons are connected to the somas of other neurons, forming a large network which is called the nervous system. Neurons are able to influence each other by way of neurotransmitter which can either be enticing or prohibiting the other neuron to 'fire'[13].

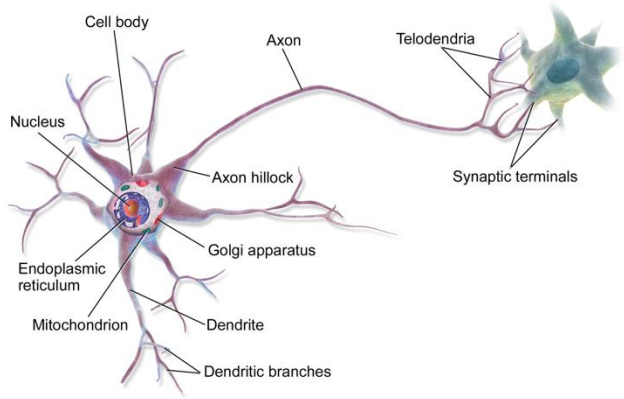


Figure 1: A biological neuron

[10] Imperial College London. (n.d.). Neural Networks. Retrieved on 11 november 2018, from https://www.doc.ic.ac.uk/%7End/surprise_96/journal/vol4/cs11/report.html
[11] Wikipedia contributors. (2018, 11 november). History of neurology and neurosurgery - Wikipedia. Retrieved on 11 november 2018, from https://en.wikipedia.org/wiki/History_of_neurology_and_neurosurgery
[12] Bijterbosch, J., & from Wijk, P. (2015). *Nectar 3e editie biologie 5 VWO leerboek* (3e ed.). Groningen/Houten, the Netherlands: Noordhoff.
[13] Bijterbosch, J., & from Wijk, P. (2015). *Nectar 3e editie biologie 5 VWO leerboek* (3e ed.). Groningen/Houten, Netherlands: Noordhoff.

An artificial neuron functions in much the same way. It is able to get many inputs and compute those to a single output which it sends forward to the next neurons. More complicated artificial neurons first have their input computed by a specific value, or 'weight'[14]. This weight influences the impact every specific input has on the neuron. This weight can be any value which represents the importance of a certain input. The neuron can also have a bias. This is a threshold which can prevent a neuron from firing if the input is not above a certain level.

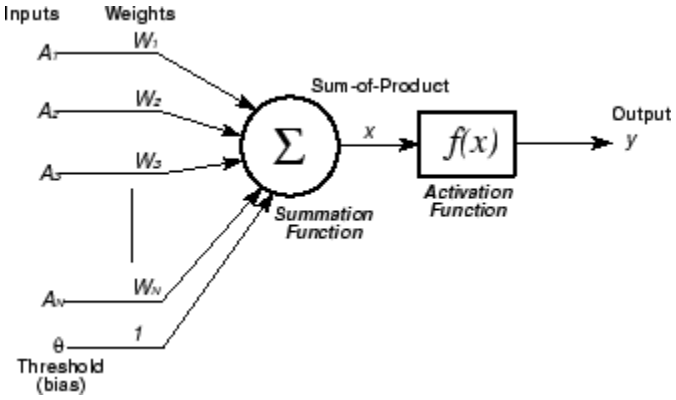


Figure 2: An artificial neuron

When combined these factors can form a mathematical equation. Below is the function for a neuron with a single input, which is computed by a weight and restrained by a bias.

$O_{(neuron)} = W_1 \cdot X_1 - b$

In which $O_{(neuron)}$ is the output of the neuron, W_1 the weight of input X_1 and b the bias.

Neurons which have an amount of inputs n have a similar, though larger, function.

$O_{(neuron)} = W_1 \cdot X_1 + W_2 \cdot X_2 + \ldots + W_n \cdot X_n - b$

[14] Imperial College London. (n.d.). Neural Networks. Retrieved on 12 november 2018, from https://www.doc.ic.ac.uk/%7End/surprise_96/journal/vol4/cs11/report.html

5.4.1 Activation functions

The value of $O_{(neuron)}$ is linear. If we increase X_1 by a certain amount the output would react to this in a linear way. Consequently, the neuron can only represent linear data[15]. In order to make the neuron non-linear we use something called an activation function. The output of a neuron can be seen as a single, non-parabolic function, y = ax + b. Different activation functions exist, each doing a different type of computation with the same idea, to make the output non-linear.

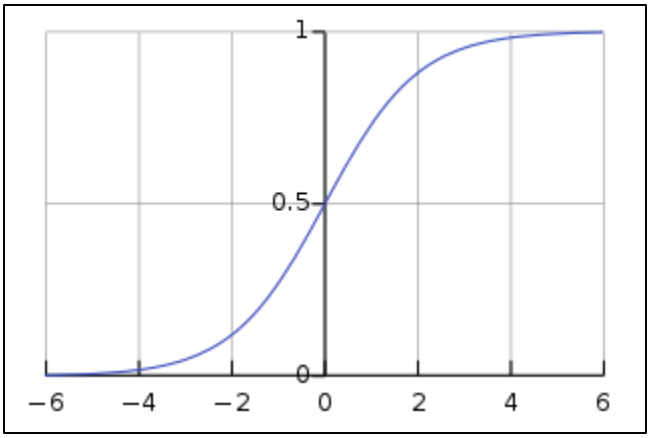


Figure 3: A sigmoid graph

Take for example the sigmoid function. Sigmoid converts a function in such a way that all y-values are 'squished' between 0 and 1, resulting in a graph like the one in figure 3.

As can be seen all possible outputs have converted onto a non-linear function, with limits x--> ∞ at y = 1 and x--> -∞ at y = 0. This function is very useful for probabilities, as it gives a certainty between 0 and 1.

A different version of the sigmoid exists, called the tanh function. This function is similar to sigmoid, but instead it has a range of –1 to 1. The advantage of using tanh over sigmoid is that outputs can have negative, or 'prohibitive', properties (biological nervous systems have these properties as well[16]).

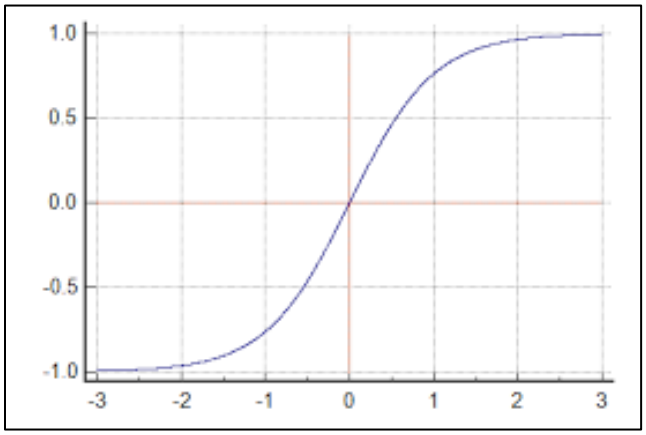


Figure 4: A tanh graph

[15] Walia, A. (2018, 24 august). Activation functions and it's types-Which is better? Retrieved on 5 december 2018, from https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f?gi=2dde68909f4dhttps://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[16] Bijterbosch, J., & from Wijk, P. (2015). *Nectar 3e editie biologie 5 VWO leerboek* (3e ed.). Groningen/Houten, Netherlands: Noordhoff.

However, most deep learning programs today use the Rectified Linear Unit, or ReLU, activation function. This function makes every negative value equal to zero and keeps all positive values proportional to their original value, resulting in a broken function like the one shown in figure 5.

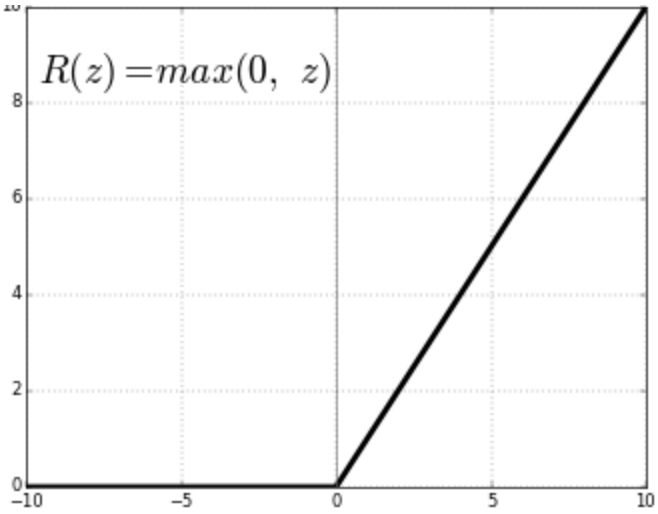


Figure 5: A ReLU graph

The ReLU function has a slight problem. As all negative values are set to zero, some neurons in the network will be 'turned off' when training. This decreases the 'capacity' of the network, as not all of its potential is used[17]. To counteract this phenomenon an even newer function has been designed, the softplus function. This function is similar to the ReLU function, with a single difference. Instead of making all negative values zero, the softplus function has a small margin left of the y-axis. The difference is shown in figure 6.

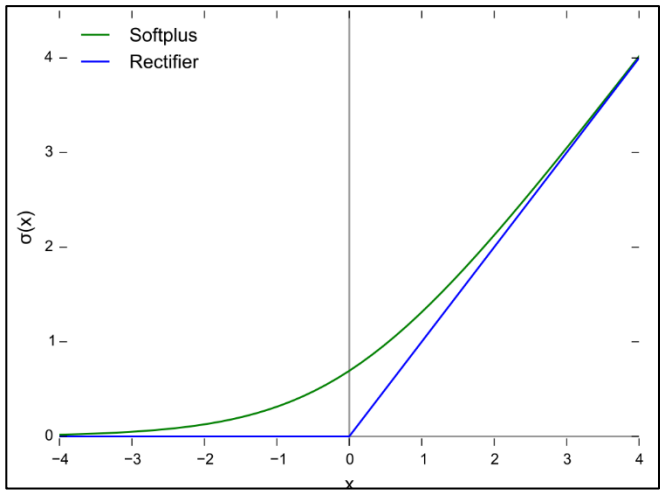


Figure 6: A softplus graph and a ReLU graph

The ReLU and the softplus functions are the most popular activation functions at this time[18]. This does not mean that the sigmoid and tanh functions are completely neglected. Both functions are great for calculating probabilities. Sigmoid gives a value between 0 and 1 for a single neuron.

In neural networks (which will be explained at a further stage in this paper) there can be multiple possible outputs. The sigmoid function is ideal for a classifier, as it gives a probability of likelihood. There is one slight problem with using sigmoid when having multiple outputs. The sigmoid function does not take into account the outputs of other neurons. Thus it could be possible that multiple classes are given a very high probability. The

[17] Walia, A. (2018, 24 august). Activation functions and it's types-Which is better? Retrieved on 5 december 2018, from https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f?gi=2dde68909f4dhttps://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

[18] Sharma, A. (2018, 21 june). Understanding Activation Functions in Neural Networks. Retrieved on 5 december 2018, from https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

softmax function has been created to counteract this. This activation function is the same as the sigmoid function, with the difference that all outputs must add up to one, thereby cancelling out the possibility for multiple highly positive outputs[19].

Regardless of the differences between activation functions, the final function for the output of a neuron will always look the same. By placing the activation in our previously created neuron we are left with our final function.

$O_{(neuron)} = Activation(W_1 * X_1 + W_2 * X_2 + \dots + W_n * X_n - b)$

5.4.2 Learning

Weights and biases cannot just have any given value. In order to correctly represent data these variables need to have specific values. Defining the optimal set of weights and biases for a neuron is how the neuron 'learns'.

A neuron starts with a random set of weights and biases. The input is then run through the neuron, which computes an output (this process is known as forward propagation[20]). This output is then compared to what the actual result ought to be. This is the so called 'loss' of the function (notice that this is characteristic of supervised learning). As the cost is the difference of the neuron's output and the actual value, we want this to be zero. The output is defined by the weights and biases (given a constant input), thus these will have to be adjusted according to the loss. Next, the loss is run back through the network, where the loss is divided over the neurons according to their influence in the output, an input which has a very large effect on the neuron will get appropriately more adjusted. This process is called backpropagation[21].

In order to compute how much each weight and bias should be adjusted the mathematical principle called gradient descent is used. The loss, or 'cost', a neuron gives can be viewed as a function, in which the input is the variable[22]. For each value of the input, the loss of the neuron has a specific value, which we want to be zero. As the input can only be directly influenced by changing its weight, the weight is the value that will be adjusted for optimal results. In order to find out in which way the weight has to be adjusted, the derivative of the function is computed. In a two-dimensional function this derivative can be either positive or negative. When the slope

[19] Multi-Class Neural Networks: Softmax | Machine Learning Crash Course | Google Developers. (2018, 1 october). Retrieved on 5 december 2018, from https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

[20] Torres, J. (2018, 26 september). How Do Artificial Neural Networks Learn? Retrieved on 12 november 2018, from https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7?gi=8b0f53201f9

[21] Torres, J. (2018, 26 september). How Do Artificial Neural Networks Learn? Retrieved on 12 november 2018, from https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7?gi=8b0f53201f9

[22] Google. (2018, 1 october). Reducing Loss: Gradient Descent | Machine Learning Crash Course | Google Developers. Retrieved on 12 november 2018, from https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent

of the function is *negative*, it means the minimum will be to its right, consequently the weight will be *increased* and vice versa.

The weight is then adjusted by a small step (called the learning rate) towards the direction of this 'local minimum'. Note: this local minimum is not necessarily the lowest cost, rather the nearest low point in the function, therefore with each new set of random weights and biases a new local minimum can be reached, yielding new results[23].

5.4.3 Gradient descent

The same principle is used when dealing with multiple inputs. In the case of a multi-dimensional cost function (which, depending on the size of the network, can range in the thousands) the derivative is not 'negative' or 'positive'. Rather, movement in each dimension will have its own value. (A two dimensional function has a direction value 'left' or right', whereas a three dimensional function will have a direction value of 'North' or 'South' and 'East' or 'West', each in different amounts. This is further clarified by the images below) The *gradient* of a multi-dimensional function is the way of steepest ascent. Thus, taking the negative of this will give the way of steepest descent. Following the negative gradient and calculating the new derivative after each step eventually the loss will reach a local minimum in much the same way as a two-dimensional function[24].

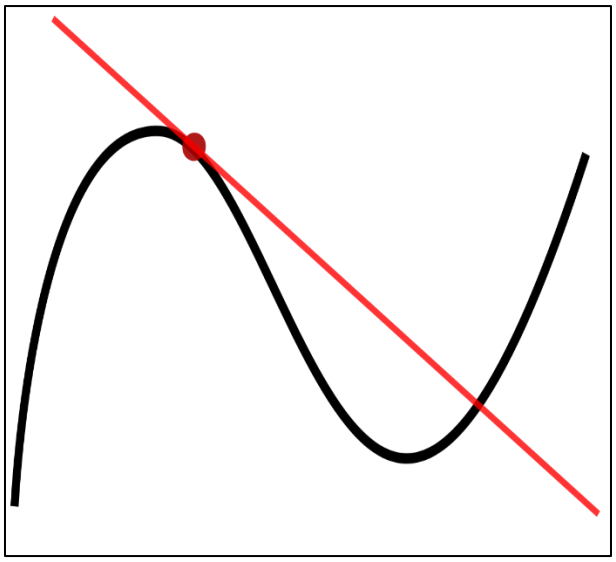


Figure 7: The gradient of a two-dimensional function

The figures 7 and 8 illustrate the difference between the gradient of a two dimensional function (X and Y) and a three dimensional function (X, Y and Z).

In order to keep the adjusting from 'overshooting' the local minimum (having the adjustment too large and ending up on the 'other side' of the local minimum) the learning rate, the amount by which the adjustment is made, can be changed according to the gradient itself. A steep

[23] Google. (2018, 1 october). Reducing Loss: Learning Rate | Machine Learning Crash Course | Google Developers. Retrieved on 12 november 2018, from https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate

[24] 3Blue1Brown. (2017, 16 october). Gradient descent, how neural networks learn | Deep learning, chapter 2 [Video]. Retrieved on 12 november 2018, from https://www.youtube.com/watch?v=IHZwWFHWa-w

gradient will have a high learning rate, whereas a gentle slope will have a small learning rate[25]. This means the closer you are to the minimum, the smaller the step that will be taken in order to prevent overshooting.

Mathematically a simple version of this can be represented by the following function.

$A_{n+1} = A_n - \alpha \cdot (J_{(A)}')$

In which A_n is the old position, A_{n+1} the new position, α the learning rate and $J'_{(A)}$ the derivative of the cost function.

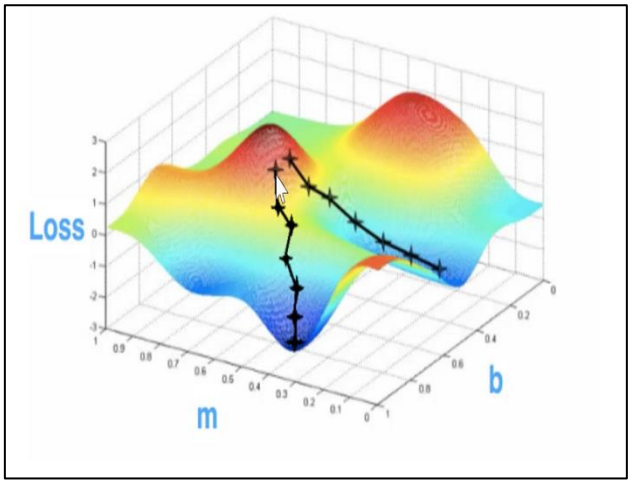


Figure 8: The gradient of a three-dimensional function

This algorithm is repeated until the learning does not progress any further. This happens when either the local minimum or the limit of the learning rate has been reached. This limit can be set in place in order to prevent the algorithm from forever taking ever smaller steps towards the minimum without ever reaching it, resulting in an infinite loop.

When the learning has been completed, the weights and biases have been arranged in what the computer has derived to be the best set of values to correctly perform the designated task. This occurrence is called 'convergence'[26]. This minimum will be different every time the network is reconfigured (the weights and biases have been reinitiated and derived). Therefore, the final score the program will achieves will differ every time.

5.4.4 Networks

When a program has multiple inputs, converging all the inputs into a single neuron which computes one output is not very effective. Why not have the outcome depend on multiple neurons, which then compute an output? In other words, why not create a network of neurons in order to better represent large data? This is the principle used when creating neural networks.

A neural network is again inspired by the human brain. In an artificial neural network, or ANN, many neurons are interconnected in different layers in order to compute outputs. ANN's are

[25] Google. (2018, 1 october). Reducing Loss: Learning Rate | Machine Learning Crash Course | Google Developers. Retrieved on 12 november 2018, from https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate

[26] Baras, J., & LaVigna, A. (1990, december). Convergence of a neural network classifier. Retrieved on 9 december 2018, from https://pdfs.semanticscholar.org/b263/9046c7d99e99119a2569fe58cedd75b000d6.pdf?_ga=2.7215792.38168314.1544376574-1239280478.1544376574

structured in three layers. The input, hidden and output layers. The input layer contains a certain number of neurons (this amount has been specified as what humans think to be the best for their input). Each of these neurons gets a single input. This input is then forwarded to the next layer, where a single neuron of the first layer sends a weighted sum of its output to each neuron of the second layer.

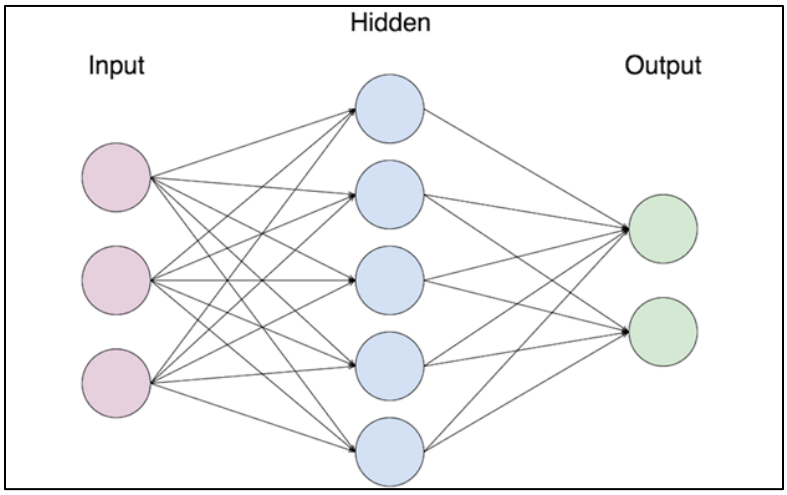


Figure 9: A neural network with one hidden layer

For example, when having 100 input neurons and 200 neurons in the hidden layer, an input neuron will send its output towards all 200 neurons in the hidden layer, each with a different weight. Thus, a neuron in the hidden layer will receive 100 different inputs, with which it computes an output. After this, each neuron in the hidden layer will forward its output towards each of the output neurons, each connection again having a different weight. The output neurons will then represent the output according to the predictions of the neural net.

Perhaps the text above is difficult to understand. In order to get a better idea of how a neural network is structured, the architecture has been visualized below.

However, some data will require even more layers of neurons in order to be able to compute accurate results. If this is the case, more hidden layers can be added to the network. The first hidden layer will then forward its weighted output to the next hidden layer et cetera. These types of networks are called Deep Neural Networks, or DNN's. (The definition on Wikipedia is as follows: "A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers.")[27]. These architectures follow the same principle as ANN's with one hidden layer. For better understanding there is, again, a picture below to visualize how DNN's are structured.

These deep neural networks can become very large when dealing with large or complex data. Consequently, training these nets will become progressively more difficult, as there will be hundreds or even thousands of weights and biases. Every single one of these values will be tweaked when training. A larger network will take more iterations to effectively train, as there are more connections to be tweaked. In addition to this, the gradient will become harder to

[27] Wikipedia contributors. (2018, 15 november). Deep learning - Wikipedia. Retrieved on 16 november 2018, from https://en.wikipedia.org/wiki/Deep_learning

compute the more inputs are added, as the gradient will be in more dimensions. Large neural networks therefore become computationally expensive to train.

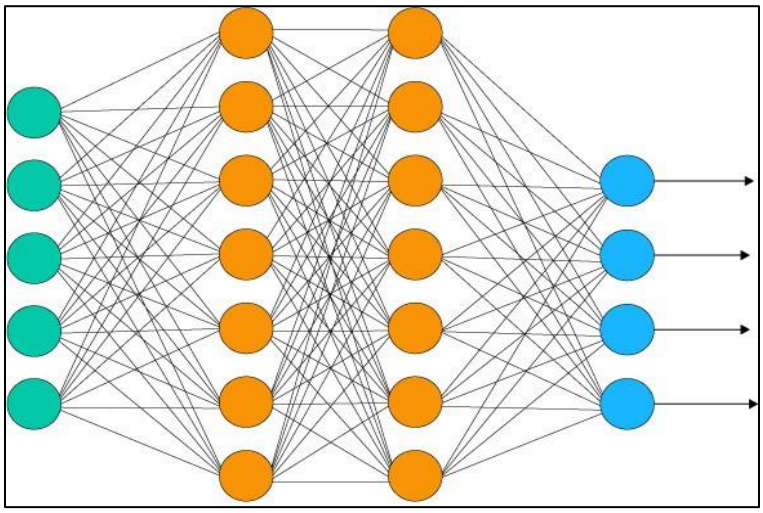


Figure 10: A neural network with two hidden layers.

Solutions to this are presented by way of batch-oriented gradient descent. The gradient descent that has been discussed op to this point is called 'stochastic gradient descent'[28]. This algorithm computes the gradient every time an input has been run through the network. However, a less computationally heavy algorithm exists, 'batch gradient descent'. This algorithm first runs all the training data through the network and then computes the gradient, taking into consideration the outputs of every input[29]. This is called an epoch. Effectively training a network will take multiple epochs. A downside of this is that all outputs will have to be stored on local memory before the gradient is computed in addition to taking a long time to compute the gradient over many outputs[30].

Training a deep neural network comes with additional difficulties. Besides the increase in training time there is a problem that can impact the quality of the network as a whole. As all neurons are interconnected between layers, neurons naturally become dependent on the input given to them by the neurons of the previous layer. As the weights of the connections change, a neuron can become more or less dependent on different neurons in the previous layer. During training a problem known as overfitting can arise. Overfitting is a phenomenon that occurs when the input of a neuron becomes largely determined by a single input[31]. This will result in the network becoming too specialized on the training data, meaning it will be less able to derive meaning from new data.

---

[28] Stanford University. (n.d.). Unsupervised Feature Learning and Deep Learning Tutorial. Retrieved on 16 november 2018, from http://deeplearning.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

[29] Brownlee, J. (2018, 27 april). A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size. Retrieved on 16 november 2018, from https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[30] Stanford University. (n.d.). Unsupervised Feature Learning and Deep Learning Tutorial. Retrieved on 16 november 2018, from http://deeplearning.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

[31] Improve Shallow Neural Network Generalization and Avoid Overfitting - MATLAB & Simulink. (n.d.). Retrieved on 5 december 2018, from https://www.mathworks.com/help/deeplearning/ug/improve-neural-network-generalization-and-avoid-overfitting.html;jsessionid=fbb36bbaaea534e816b5b7bd4b93

To prevent a network from overfitting the dropout function has been designed. Dropout is only active during training, in which it 'turns off' randomly selected neurons during the forward propagation. These neurons will consequently have no part in computing the output and thus will not be affected during backpropagation[32]. This allows the network to strengthen other neural connections thereby preventing, or greatly reducing, overfitting.

5.4.5 Size and Shape

When a network becomes larger, it is inevitable the training time increases. Therefore, it is not necessarily 'better' to make an excessively large network that will very accurately present output when configured, yet taking ages to train correctly, if at all. It is imperative to find a balance between the size of the network and the training time while still being able to compute accurate results. The number of layers in the network and the number of neurons in each layer need to be carefully chosen when creating a network.

For example, a picture of 8x8 (64 pixels) will require 64 input neurons to represent the input whereas large data structures may take hundreds of inputs. If the output is a simple yes or no (binary) answer, or a percentage, a single output neuron will suffice, whereas a classification system requires an output neuron for each possible class. In short, the number of neurons should be large enough to properly represent the data, yet not so small that the data is 'overgeneralized'.

Similarly, the number of neurons in the hidden layers, and the number of hidden layers itself ought to be chosen carefully. Simply adding more layers and neurons will not improve the network. Training time will increase and accuracy will not necessarily increase. A more complex network will require more training data[33]. More neurons may therefore lead to a less effective network. Finding the optimal number of layers and neurons is of paramount importance to a network. This optimum will differ depending on the type of problem, the number of classes, the size of the dataset and the size of the data itself. Tweaking the number of neurons and reviewing the effect it has on accuracy and training time is the only way to find this optimum.

[32] Brownlee, J. (2017, 30 march). Dropout Regularization in Deep Learning Models With Keras. Retrieved on 5 december 2018, from https://machinelearningmastery.com/dropout-regularization-deep-learning-models-Keras/

[33] Shivkumar, A. (2015, 29 october). Why would adding more neurons per hidden layer in a neural network hinder convergence within the actual training set? [Forumpost]. Retrieved on 7 december 2018, from https://www.quora.com/Why-would-adding-more-neurons-per-hidden-layer-in-a-neural-network-hinder-convergence-within-the-actual-training-set

5.5 Python

Now that that the basic theory of neural networks has been explained, the question remains: how to program them? We have done that using Python, a computer programming language designed to be easy to read and simple to implement. Although readability and ease of use are important in order to be able to focus more on the research than on the code, the biggest reason that we chose Python for our research is because it is already widely used throughout the field of Machine Learning[34]. Consequently, there are a lot of open-source libraries that can be used for creating Neural Networks such as TensorFlow, Pytorch and Theano. A library in computer programming is basically a piece of code which is written by somebody in order to make another task easier or make it require less work. If someone wanted to hang a painting on the wall, a hammer could be used to put the nail in the wall. A library is essentially a tool made by somebody in order to accomplish a task.

In order to understand the code that will be written for the voice recognition program, basic python functionality will be explained. A basic understanding of math will be very useful.

The most basic functionality of Python, and most programming languages, is the ability to store variables. A variable is like x in math, a name which can store a value. To create a variable in python, all that has to be done is:

$$variableName = value$$

The variable name has to be valid. In order to be valid, the name has to be a consecutive row of characters. In order to use several words as a variable name, camel-case is used. It means the first character of every word after the first is capitalized. That way it is easier to remember the variable name and clearer what the variable actually is. The value of the variable has to be a certain type. It can be an Integer (number), String (text) or Boolean (true or false). These are not the only variable types, however, to stay concise, the basics are enough.

What does have to be addressed is an array. An array can be declared by:

$$arrayName = [value1, value2, value3]$$

An array is a list of values. Each value within an array can be accessed individually and arrays are mostly used to store values which have some sort of relation to each other.

Next, functions are a very important feature of every programming language. Just like in math, functions can have one or multiple inputs and they can return an output. Functions are defined by:

[34] Voskoglou, C. (2017, May 5). What is the best programming language for Machine Learning? Retrieved on January 17, 2019, from https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7?gi=80bad449c44d

$$def\ functionName(argument):$$

The ‘def’ at the beginning is a keyword. Keywords in Python are used to *let the programming language know* that a certain feature is going to be used. Here it is to define a function. The arguments that are given with the function when it is called can be used within the function. Variables that are declared in a function cannot be used outside of that function, they will disappear after a function has returned or stopped. The syntax for calling a function is not much different than the way it is declared, it is quite self-explanatory:

$$functionName(argument)$$

Last but not least, a library can be needed when programming in Python. Using a library requires it to be downloaded and declared within the file of code like this:

$$import\ libraryName$$

In this case, ‘import’ is the keyword to indicate that a library needs to be imported. A library can also be renamed within the code using the keyword ‘as’ and the new name directly after the library name. It is also possible to import only certain functions from the library:

$$from\ libraryName\ import\ functionName$$

Apart from the libraries to convert the audio data and perform the Fourier Transformation (Chapter 6), Keras will be the library used for creating the model of the Neural Network. Keras is a library based on TensorFlow, a library made by Google to create Deep Neural Networks using your own data. Keras is a framework on top of TensorFlow which makes it easier to use and it requires less knowledge to start creating Neural Networks. As said before, to focus on the research rather than figuring out the code, we chose the easier option which was Keras in this case.

Even though Keras is the ‘easier’ option, it still requires decent knowledge of programming, Neural Networks and the different optimization and activation functions for optimizing or altering the Neural Network. Keras will be explained further in chapter 7.3.

5.6 Summary

In conclusion, there are many different types of AI and ways these programs learn. For our problem we have decided to make use of Artificial Neural Networks. These computational architectures are based on the most advanced form of intelligence on Earth, the human brain. Neurons take weighted inputs, subtracts a bias and forwards the output through an activation function. Neurons learn by optimizing the weights and biases, which is done using the gradient descent algorithm, an algorithm that computes the fastest route to a local minimum, an optimal combination of weights and biases.

To increase the ability of the program to handle large data structures many of these neurons can be arranged in a Deep Neural Network. These networks consist of an input layer, hidden layers and an output layer. The way this network is given form should be carefully chosen and thoroughly tested. The optimal structure is not always the largest, but the one that best encapsulates the data it takes in, the way it is computed and the output it presents.

Finally, in order to create the actual Neural Network in a program we will be using a computer programming language called Python. To build the Neural Network we are going to use Keras, a framework based on the popular Machine Learning library TensorFlow. It is relatively easy to use in comparison with TensorFlow, thereby leaving us more time to focus on the research instead of writing code.

6. Voice Recognition

In this chapter we will dive deeper in the subject of voices. We will explain what they are made up of, how we can use them for recognizing speakers and how we can transform the voices so that they can be used as input for the neural network.

6.1 Voices in general

The human body is capable of creating many different sounds. When a human says a word, the air in the lungs is let out under pressure through the vocal cords. These bits of tissue then vibrate due to the air pressure. This vibration is amplified in different aspects by using the muscles to change the tension and shape of these cords. The sound is then led through the mouth, where the cheeks, tongue, jaw and lips will 'finish' the sound produced by the vocal cords[35].

When using these factors in different ways, different sounds can be produced. Humans can combine these different sounds into words, and words into sentences. Each word has a definitive sound. This sound is created by a 'preset' of stances in the vocal cords and mouth. This preset is similar in every human being. However, each person has a certain way it produces these sounds. This can be due to a number of reasons. The physiology is different with each individual and every person has these 'presets' configured in a different way. In other words, nature and nurture both affect the sound of your voice, as stated by the university of Iowa[36].

But what is a voice made up of? When a person says the letter 'E', the vocal cords and stance of the mouth produce a sound which is made up of multiple frequencies. These frequencies are combined into a single conjoined function, or continuous sound. Each person's 'E' is unique, as it is made up of different frequencies[37]. Consequently, if these frequencies could be 'untangled', each person will have a different set of frequencies, thus each person can be identified according to their voice pattern.

6.2 Difficulties

When a soundwave is recorded by a microphone, the wave hits a membrane which oscillates according to the sound wave. This vibration is then picked up by the microphone because an

[35] Wikipedia contributors. (2018, 16 november). Human voice - Wikipedia. Retrieved on 16 november 2018, from https://en.wikipedia.org/wiki/Human_voice

[36] University of Iowa. (n.d.). Nature versus nurture of voice | voice-academy. Retrieved on 16 november 2018, from https://uiowa.edu/voice-academy/nature-versus-nurture-voice

[37] McLachlan, H. (2016, August 11). Is every human voice and fingerprint really unique? Retrieved on January 20, 2019, from https://theconversation.com/is-every-human-voice-and-fingerprint-really-unique-63739?sr=1

attached magnet is moved nearer to, or farther from, a coil[38]. Consequently, the only thing a microphone records is the frequency over time (along with the intensity of the sound).

The recording of a voice is thus the combination of all formants by which it is made up. As stated before, these formants are different for each person, thus the formants that a tone consists of are different. Identifying people by their voice can therefore be done by finding thee formants the tone is made up of.

In mathematics the Fourier transform can be used to achieve this. When having a combined sinus wave, a function can become very complicated, as illustrated below.

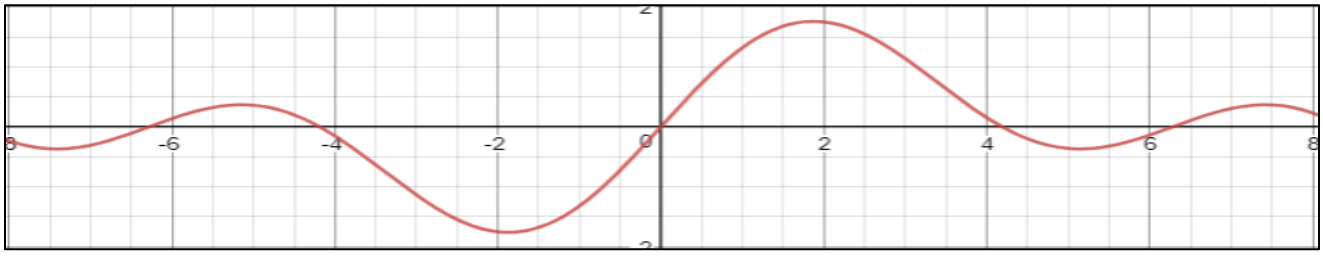


Figure 11: A graph of the function f(x) = sin(x) + sin(0.5x)

Looking at this function, a clear symmetry can be seen, though the function itself looks random. In reality this function is just $sin(x) + sin(0.5x)$, a simple combined sinus function.

Both sin(x) (red) and sin(0.5x) (blue) are shown below.

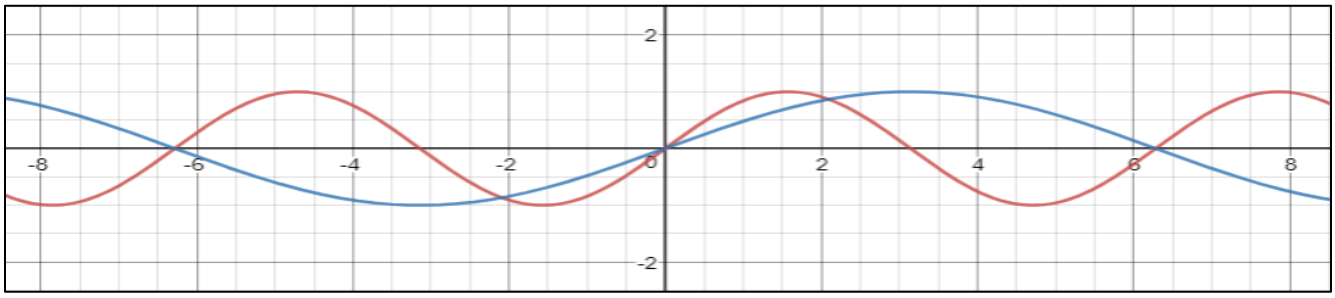


Figure 12: The graphs of sin(x) (blue) and sin(0.5x) (red)

The combined function is nothing more than the added values of the red and blue lines shown above. Sometimes these lines reinforce each other, while other times they cancel each other out. The Fourier transform is used to find the individual functions making up the combined function[39].

[38] Wikipedia contributors. (2018, 16 november). Microphone - Wikipedia. Retrieved on 17 november 2018, from https://en.wikipedia.org/wiki/Microphone

[39] Wikipedia contributors. (2018, 16 november). Fourier transform - Wikipedia. Retrieved on 17 november 2018, from https://en.wikipedia.org/wiki/Fourier_transform

6.3 Fourier transformation

The Fourier transform is rather difficult to explain without getting into complex mathematics. To explain this problem more intuitively, imagine the function cos(x), but instead of it being shown in a x/y grid, it is 'wrapped around' a single point, for example the origin. The orbital period (the time it takes to go around the circle exactly once) of this circle is variable.

The x-value of the function corresponds to the amount of rotation around the point and the y-value is the distance between the point and the function. A visualization of this is shown below.

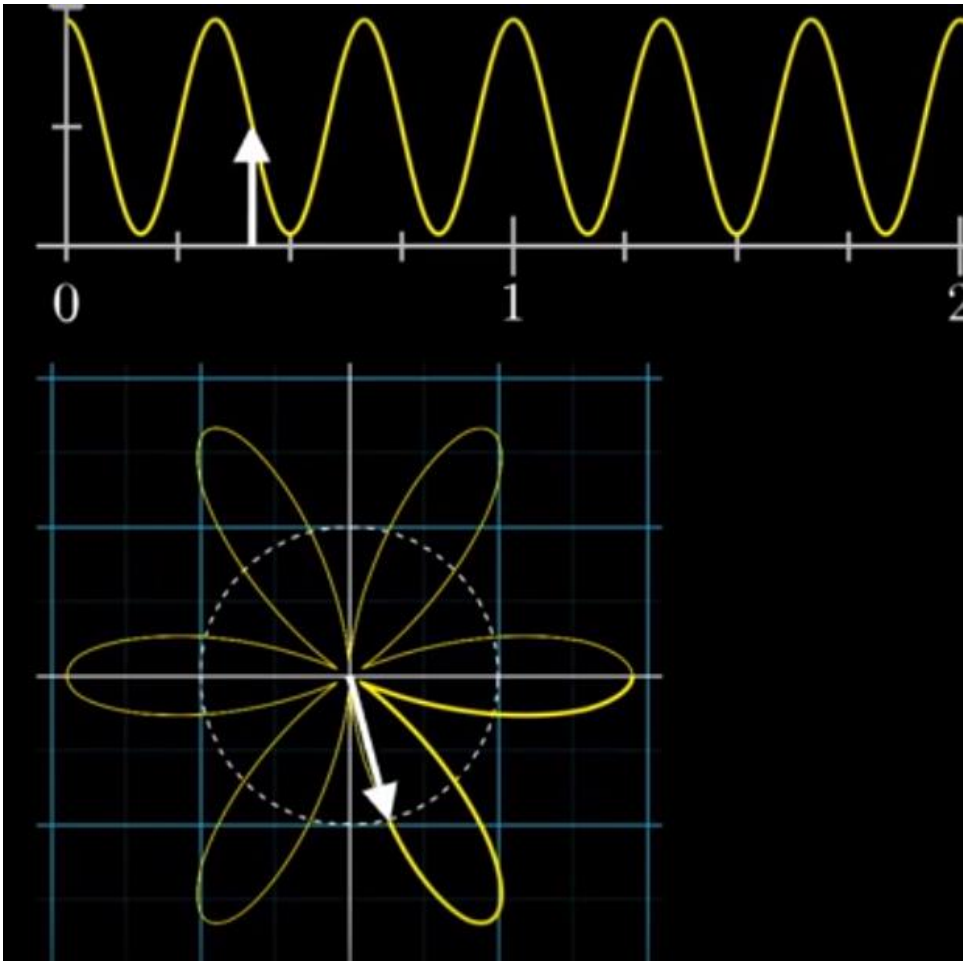


Figure 13: The graph of the function f(x) and the same function wrapped around the origin with an orbital period of 2 ([40]41

As can be seen, the orbital period of this function in this example is 2. When changing said period the shape of the circle graph will change drastically.

[40] 3Blue1Brown. (2018, 26 january). But what is the Fourier Transform? A visual introduction. [Video]. Retrieved on 18 november 2018, from https://www.youtube.com/watch?v=spUNpyF58BY

[41] 3Blue1Brown. (2018, 26 january). But what is the Fourier Transform? A visual introduction. [Video]. Retrieved on 18 november 2018, from https://www.youtube.com/watch?v=spUNpyF58BY

The orbital period can also be expressed as a frequency. The formula for frequency is 1/T, in which T is the vibration time (in this case orbital period) in seconds. This example has therefore a frequency of ½ hertz (Hz).

When setting the frequency to 3 Hz the circle will capture the exact vibration time of this cosine function, as shown in the following image[42].

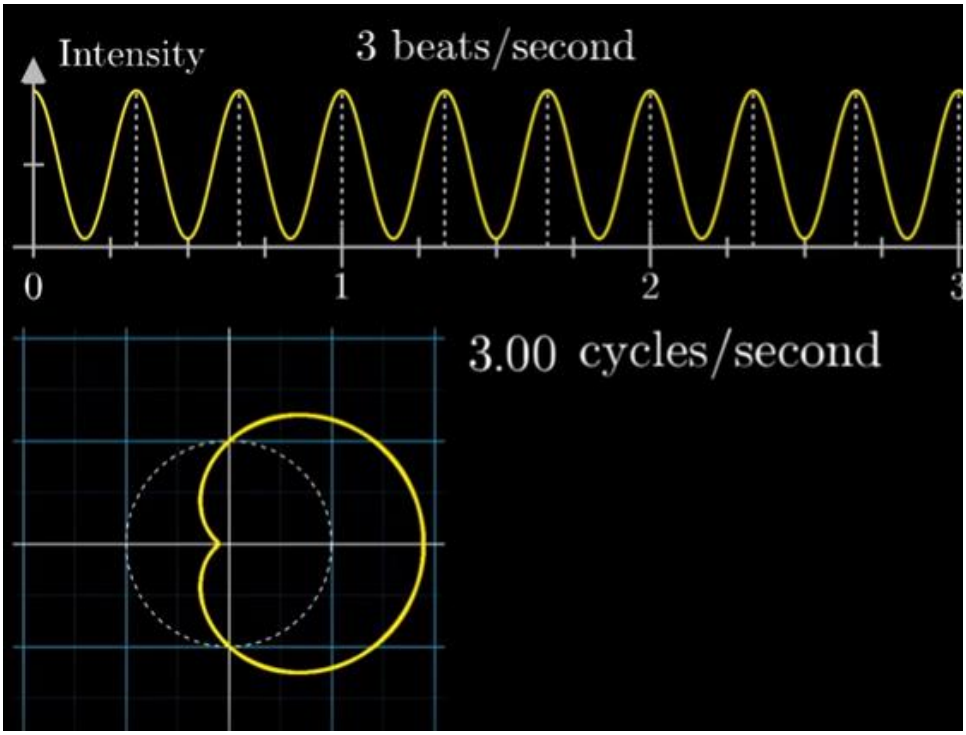


Figure 14: The same function f(x) wrapped around the origin with an orbital period of 1/3 ([43])

Imagine this graph has a 'mass' assigned to it, equally distributed over the entire function. The center of gravity of this graph would change when the rotation time of the circle is changed. The center of mass of graph 1 would be somewhere around the origin. The center of gravity would stay around the origin for every given rotation time, except for the one above, 3 Hz. In this case, it sits at (1, 0).

The x-coordinate of the center of gravity is plotted out over the winding frequency in the image below.

[42] 3Blue1Brown. (2018, 26 january). But what is the Fourier Transform? A visual introduction. [Video]. Retrieved on 18 november 2018, from https://www.youtube.com/watch?v=spUNpyF58BY
[43] 3Blue1Brown. (2018, 26 january). But what is the Fourier Transform? A visual introduction. [Video]. Retrieved on 18 november 2018, from https://www.youtube.com/watch?v=spUNpyF58BY

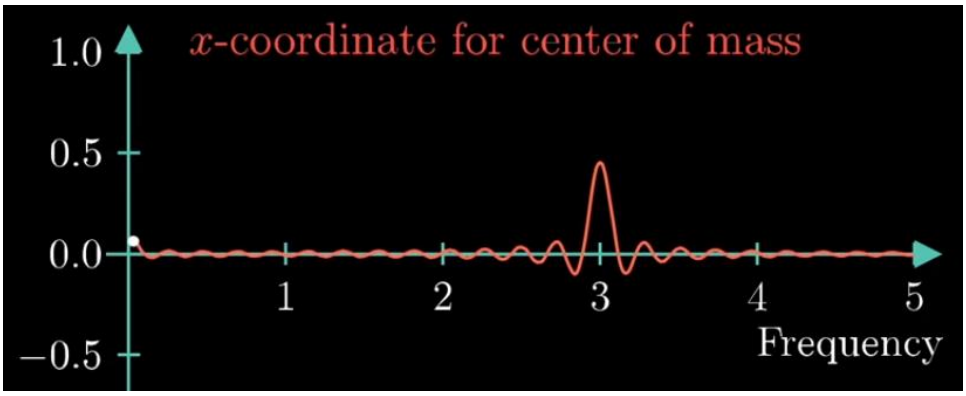


Figure 15: A graph of the x-coordinate of the center of mass as a function of the frequency ([44])

Notice there is but a single peak in this graph, at x = 3, which is the same as the frequency of the cosin function.

When having a combined cosin function, for example one with frequency 2 Hz and another with frequency 3 Hz, the center of mass will be far to the right on two occasions, one at x = 2 and another at x = 3.

However, the center of mass also is a coordinate, thus it has an associated y-value as well. With respect to the frequencies that make up a tone this is not relevant, however, when describing the Fourier transform mathematically this complex number does come into play.

However, before explaining the Fourier transform one must understand how a function is 'wrapped' around a point in general. For this, Euler's famous formula $e^{\alpha i} = \cos(\alpha) + i \cdot \sin(\alpha)$ is used[45].

According to Euler's formula a circle with radius 1 in the complex plane can be described using the function $e^{\alpha i}$. In which α is equal to the length of the part of the circle from the beginning to the current location. In order to 'wrap' the cosin function around a circle Euler's formula is used.

The function is then $e^{2\pi i f t}$. In this function, 2π describes the length of the circle and f * t describes the rotation, frequency multiplied by time. However, Fourier transforms are wrapped around the circle in a clockwise direction, thus the exponent has to be negative. Consequently the function will become $e^{-2\pi i f t}$.

This function describes a circle that rotates clockwise and has a radius of 1. When 'wrapping' a function around the origin, all that must be done is multiplying the function (f(t)) by the circle

[44] 3Blue1Brown. (2018, 26 january). But what is the Fourier Transform? A visual introduction. [Video]. Retrieved on 18 november 2018, from https://www.youtube.com/watch?v=spUNpyF58BY
[45] Intuitive understanding of Euler's formula. (n.d.). Retrieved on 19 november 2018, from https://betterexplained.com/articles/intuitive-understanding-of-eulers-formula/

function ($e^{-2\pi ift}$), giving the following function: $a(t) = f(t) \cdot e^{-2\pi ift}$, where a(x) is the wound-up function.

In order to calculate which frequencies make up the function f(x), the center of mass needs to be tracked. This can be done by taking the integral of a(x) for every value (-∞ to ∞), resulting in the function $\int_{-\infty}^{\infty}(f(t) \cdot e^{-2\pi ift})_{dt}$ [46].

The 'real' component of this function (a complex function has a real as well as an imaginary part) will show peaks at corresponding frequencies. Frequencies that persist in the combined function (f(t)) will have greater values in the output graph (the graph which displays the frequencies)[47].

Thus, when having a function $f(t) = sin(t) + sin(0.5t)$, the Fourier transform of this function will show two peaks, one at ½ · π and another at ¼ · π, which are the same frequencies the original signal was made up of. This is what the Fourier transform does, outputting the frequencies of which a signal consists. The Fourier graphs of the three speakers that have recorded input for the neural network are shown in the Attachments, section E.

6.4 Summary

In conclusion, the Fourier transform can be used to discern the frequencies a signal is made up of. The way a voice sounds is determined by both nature and nurture, leading to each person having a unique voice. A recording of the human voice can be transformed using the Fourier transform, outputting a graph that shows which frequencies make up the voice. As each person's voice is unique, each person will have a unique Fourier transform output. This 'signature graph' can be used as input for a neural network, which then should be able to differentiate between graphs, consequently differentiating people using only their voice.

[46] Fourier transform for dummies. (z.d.). Geraadpleegd op 19 november 2018, van https://math.stackexchange.com/questions/1002/Fourier-transform-for-dummies

[47] 3Blue1Brown. (2018, 26 january). But what is the Fourier Transform? A visual introduction. [Video]. Retrieved on 18 november 2018, from https://www.youtube.com/watch?v=spUNpyF58BY

7. Research

In this chapter we will go deeper in to the research we have conducted, going through the process step by step. We will explain the recording of the voices and how we have programmed the Fourier transform and neural network in python in addition to the steps we have taken to optimize the networks.

7.1 Recording voices

Before individuals can be recognized using their voice, it, obviously, has to be recorded. Naturally the recording has to be high quality as it is of paramount importance to this research. Aside from a high quality microphone a few rules have been set in order to generalize the way audio is recorded. In a way this can be seen as a protocol that has to be followed in order to have valid recordings for the research.

First of all, every audio sample is recorded on the same microphone. Each microphone has a different design, therefore using one microphone will yield reliable results in respect to the quality of the recording.

Secondly, the distance between the mouth and the microphone might influence the recording, thus we have standardized this as well. As a rule of thumb, the speaker has to make a pistol shape with his or her fingers and place the thumb on the point of the chin. The index finger should then exactly touch the microphone. This will create about 9 to 10 centimeters of distance between the mouth and the recorder.

Thirdly, it is important there are no fluctuations in the recording. When recording, the vowel must be said in a single, monotone pitch. When saying ‘E’, humans start with somewhat of a boost. The first ½ second of saying a vowel there is some kind of emphasis. It could be possible this emphasis could influence the recording in unforeseen ways. To prevent this the speaker has to start speaking before the recording starts, about 1 to 1½ seconds prior. In addition to this the pitch of the recording has to be uniform for all samples, as the Fourier transform of high pitch samples was much unlike the lower pitched samples.

Moreover, at the end of a sentence, humans tend to ‘lose power’, their voice dies down. As with the ‘boost’ before the recording, the change of pitch this phenomenon has may influence the results. Similarly, as the beginning is excluded from the input, the end is as well. The recording ends before the speaker stops speaking.

Lastly, in order to restrain the impact of cracks and vibrations in the voice the length of the recording has to be limited. The longer a voice continues, the harder we found it to keep our pitch stable. Therefore we decided to standardize the length of the recording to three seconds.

The recorded voice is then edited using Audacity. Audacity is recording software which we use for 'cutting' the samples to the right length. After the voice is properly recorded and trimmed it is saved as a .wav file. The reason this file format has been chosen over the likes of .mp4 is because of the next stage in the research, which we will get to shortly.

We set these regulations for recording training data in order to maintain a high, and uniform, audio quality. As our goal is to achieve over 90% accuracy on the network, good audio is important, for it is the basis the entire research relies on.

7.2 Transforming voices

When the audio samples have been recorded and organized, they have to be converted before used as input our program can use. The 'frequency/time' file has to become a 'frequency/intensity' file, meaning the audio will have to be converted using the Fourier transform.

Using python we made a program that can perform the Fourier transform on a .wav file. To properly explain how this has been achieved the code will be explained line by line below. Some variables may have 'temp' in their variable names. This is short for temporary and is a way to say that variable is temporarily used and will probably be discarded.

```
import matplotlib.pyplot as plt
import matplotlib.axes as ax
from scipy.io import wavfile as wav
from scipy.fftpack import fft
from pathlib import Path
import numpy as np
import wave
import sys
import pickle
```

This first bit of code is merely importing the different libraries and packages necessary for converting the audio files. The next bits of code are far more interesting.

```
def convert_train_data(audioVowel):
    global fileName1
    global fileName2
```

This piece of code simply puts the code into a function with 'def', which makes it easy for us to input a vowel like 'E' or 'A' and automate the process. We're using global filenames here which we did so that we could enter the key variables all in one place. If, for example, we would want to use another file, we do not have to change code all over the place but change it in just one place.

```
    # Train Data
    labelData = []
    inputValues = []
    print('Steven train data')
    inputValues, labels = convert_train_data_from(fileName1, audioVowel,
inputValues)
    labelData.extend(labels)
    print('Matthijs train data')
    inputValues, labels = convert_train_data_from(fileName2, audioVowel,
inputValues)
    labelData.extend(labels)
```

Here another function (convert_train_data_from()) is called twice to operate on both Steven's and Matthijs' training files. This data is stored in a large list.

```
    # Test data
    print('Test data')
    labelData_test = []
    inputValues_test = []
    inputValues_test, labels = convert_train_data_from(fileName1 + 'Test',
audioVowel, inputValues_test)
    labelData_test.extend(labels)

    inputValues_test, labels = convert_train_data_from(fileName2 + 'Test',
audioVowel, inputValues_test)
    labelData_test.extend(labels)

    return np.array(inputValues), np.array(labelData),
np.array(inputValues_test), np.array(labelData_test)
```

This piece of code does essentially the same as the previous one except it now calls the function on the test data of Steven and Matthijs. The test data is used to determine the accuracy of the neural net later.

```
def convert_train_data_from(audioFileName, vowel, data_list):
    global fileName2
    x = 1
    temp_labels = []
```

In the second function the actual Fourier transformation takes place. The first two parameters of the second function are quite self-explanatory. The third parameter is a list of variables, which the data that has been created with the Fourier transformation should be appended to. The fileName2 variable is to check which file it is creating data of; the x variable is to simplify transforming the data of all the audio files that are currently present instead.

```
    while True:

        audiofile = '../AudioFiles/'+ str(vowel) + str(audioFileName) +
str(x) + '.wav'
        if Path(audiofile).is_file():
            spf = wave.open(audiofile,'r')
        else:
            break

        print(x)
```

The first line in this piece of code creates a while loop, this way with the next couple lines the code keeps doing the transformation until there are no more new audio files available. This does require all of the audiofiles to be named in the same way. In this case we used the format: [Vowel][name][number] so for instance 'Ematthijs1'.
The x that was declared earlier is used here so in a later piece of code it adds 1 to x, so that it checks every number from 1 to n until there is no file with that name anymore. Next, the if statement checks if the file exists, if so, it opens that audiofile. If the file does not exist, it stops the while loop. The 'print x' statement is to keep track of where the conversion currently is, since it takes quite some time to transform all audio files.

```
        #Extract Raw Audio from Wav File
        signal = spf.readframes(-1)
        signal = np.fromstring(signal, 'Int16')
        fs = spf.getframerate()
```

Now the frames or "signal" from the file is read and is converted into a type of 'Int16', basically a number that has a minimum and maximum which it can store (-32,768 to +32,767). Next the framerate is stored in the variable fs.

```
        # Time and fft
        fft_out = fft(signal)
        xvalues=np.linspace(0, len(signal)/fs, num=len(signal))
```

Now the Fourier transformation is performed on the signal with the function fft() which was imported earlier from the 'scipy' library. In the next line a variable xvalues is made. This variable is an array of numbers which if plotted, would equal the time of the recording on the x-axis.

```
        # Get corresponding y values
        yvalues = np.abs(fft_out)
        framerPerSecond = int(181855 / 4)
```

Here the yvalues is an absolute array of the Fourier transformation output. The 'framerPerSecond' is a fixed number which is necessary for the next piece of code to work properly, it is the amount of frames that is equal to one second.

```
# Create new array for new graph of values
        yArrayValues = []
        for i in range(0, int(framerPerSecond*0.14), 8):
            idx = np.where(xvalues==xvalues[i])
            yArrayValues.extend(yvalues[idx]/(1* (10**7)))
            if i >= 3996:
                break
```

First a list is created here. This list will represent a certain amount of the y values to make the input for the neural network smaller. What the for loop here does, is it goes through a range of numbers by steps of 8. The range (framerPerSecond*0.14) is a maximum of where y-values of worth could be. It makes an array of these values in yArrayValues and it also divides by a fixed number (10**7 which means 10^7) to make the values smaller so it's generally easier to work with. The i >= 3996 makes it only get 501 values from the array. This was made so it is easily adjustable how many neurons we would actually want to use in the Keras code.

```
        z=0
        cumulativeValue = 0
        zArrayValues = []
        for value in yArrayValues:
            cumulativeValue += value
            if z == 10:
                zArrayValues.append(cumulativeValue/10)
                cumulativeValue = 0
                z = 0
            z += 1
```

Now this part of the code is for neural network 2, not 1. Globally, this part of the code takes the average of every 10 values of the yArrayValues and puts it in a new array zArrayValues. The reason for this is the reduced amount of neurons, which will be explained more later.

```
        data_list.append(zArrayValues)
        if audioFileName == fileName2:
            temp_labels.append([1, 0])
        else:
            temp_labels.append([0, 1])
        x += 1
    return data_list, temp_labels
```

This bit of code of the second function first adds the zArrayValues (so the neurons for one file) to the data_list, which is a list that consists of these lists of neurons. Next, if the current audiofiles are that of fileName2 (so Matthijs) an array will be appended with [1, 0]. This is a label for the neural network so if it is training on a particular file, it tells the neural network "this is Matthijs" and if it guessed that wrong, it will improve. Vice versa if it appends [0, 1]. Finally, the function end with a line that is less indented because it is not in the while loop. This will return the data if all the files have been converted.

```
fileName1 = 'steven'
fileName2 = 'matthijs'

z = 1
storeFile = 'filename'
while True:
    if not Path(storeFile + str(z) + '.pckl').is_file():
        storeFile = storeFile + str(z)
        break
    z += 1

f = open(storeFile + '.pckl', 'wb')
pickle.dump(convert_train_data('E'), f)
f.close()
```

This last piece of code, outside of any function, sets the global variables (the two filenames) and makes a file with a .pckl extension so it's easy to get the stored variables in another python program. The while loop in this piece of code is mostly to make it easy to store files. It checks if the current filename, from the 'storeFile' variable, already exists. If it does then it adds a 1 to the filename and checks again if it exists. This way you don't continually have to change the file name if the program has to run multiple times.

The code can also output a graph of the performed Fourier transform. Six of these graphs are shown in the Attachments (section E).

7.3 Creating the neural net

The conversion file transforms the audio into an array of 50 different values. This array represents the Fourier graph by dividing the it in 50 equal pieces, of which the average y-value was used as input. The values of the array are the y-averages of the function, the position in the array represents the number of the division along the x-axis. The array serves as the input for the neural network.

Programming a neural network from scratch, meaning programming every neuron individually, would take an astronomical amount of time when dealing with a network of this size. Each neuron, the connections between neurons and their respective learning algorithms would have to be programmed one by one. A machine learning library called Tensorflow can be used to speed up the process. Using Tensorflow layers of neurons and their 'characteristics' such as optimizers, architectures and activation functions can be created with but a few lines of code.

However, Tensorflow's many possibilities can make it hard and confusing to use. Therefore we have decided to use Keras, an even higher level api build on Tensorflow that is created specifically for programming neural networks. Using Keras neural networks can be quickly created and easily modified. In addition to this the code is shorter and clearer compared to using Tensorflow.

The code below is that of the SENNSORS network itself, which is the actual neural network that trains and predicts on the transformed audio input.

```
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras import optimizers
import pickle

from scipy.io import wavfile as wav
from scipy.fftpack import fft
from pathlib import Path
import numpy as np
import wave
import sys
```

The first bit of code is for importing the different libraries we will need for the program, including the different Keras models required for creating the network.

```
def tensorModel(output_neurons, output_activation_func, loss_calculator,
trainFile):
    global x_train
    global y_train
    global x_test
    global y_test
    global x_individual
    neurons = 50
    activation_func = 'relu'
```

The tensorModel function is the function in which we will define our neural network. It takes 4 inputs, three of which, the variables output_neurons, output_activation_func and loss_calculator are in place for the sake of keeping the network flexible, as will be explained later. The last input, trainFile, is the most important for the network, namely the converted audio files.

The first part of the function defines the variables that will be used. The second part of the function is far more important.

```
    model = Sequential()
    # Input layer
    model.add(Dense(neurons, kernel_initializer="uniform", input_dim=neurons,
activation=activation_func))

    # Hidden layers
    model.add(Dense(neurons, kernel_initializer="uniform",
activation=activation_func))

    model.add(Dense(neurons, kernel_initializer="uniform",
activation=activation_func))
    model.add(Dropout(0.5))
```

```
    model.add(Dense(neurons, kernel_initializer="uniform",
activation=activation_func))
    model.add(Dropout(0.2))

    # Output layer
    model.add(Dense(output_neurons, kernel_initializer="uniform",
activation=output_activation_func))
    adam = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None,
decay=0.0, amsgrad=False)
    model.compile(loss=loss_calculator, optimizer=adam)
```

This bit of code is the neural network. A general overview shows a few things. First of all the code is very clear and easy to edit. This is due to our use of Keras. Secondly the structure of the network can be seen almost immediately. There is one input layer with length “neurons”, which has been defined as 50 (the length of the input array) in the prior bit of code. There are three hidden layers, all of which have length “neurons” as well and there is one output layer with length “output_neurons”, that will be defined when the function is called.

The first line of code is “model = Sequential()”. This tells Keras the architecture of the network that will be created. There are multiple different architectures, sequential is the one that has been described earlier in this paper and that we have found to be the best architecture for our problem.

The next few lines of code look very similar. To define a layer of neurons using Keras there is a ‘template’ that has to be filled in. This template can be described as follows:

$$model.add(layer(neurons, initializer, activation\ function))$$

The ‘layer’ variable is in our network ‘Dense’. A dense layer in Keras is a traditional, fully interconnected neuron. Other types of layers can be added, but we found a classical neural network more suited for our research.

‘Neurons’ defines the amount of neurons in the layer, in our case it is defined as the previously stated variable ‘neurons’, set as 50.

The ‘initializer’ variable defines the start values of the weights and biases. In our model it is ‘uniform’, which makes the starting values of the weights and biases the same over the entire layer.

The last variable is ‘activation function’, which speaks for itself. The input- and hidden layers have a ReLU activation function, as this is widely agreed to be the best choice in most situations regarding neural networks. However, in the output layer we use different activation functions depending on what input we give the network. Later in this chapter we will explain this topic further.

Below the second and third hidden layers there is an additional line of code, defining a variable called ‘Dropout’. In the chapter 5, Artificial Intelligence the term dropout has already been

explained. In short, it is a means to prevent a network from overfitting neurons. The dropout function has a value which defines the amount of dropout in each layer. Looking at the code it is notable that the first layer does not have dropout, the second layer has a high (0,5) dropout rate and the third layer has a low (0,2) dropout rate. The reason the dropout is adjusted in this way is simple. During testing this configuration (along with only using three hidden layers) yielded the most consistent and accurate results.

Below the output layer is a variable called 'Adam'. Adam is a deep learning optimizer algorithm pre-programmed in Keras. There is a long list of possible optimizers available in Keras, but we have chosen to use Adam for two reasons. First of all it seems to be the most popular optimizer to use at the moment, but more importantly this optimizer gave far better training results than any other optimizer we have tried to use.

Keras has a standard Adam optimizer, but it also has the possibility to configure the settings yourself by using the optimizers library. Between the parentheses there are multiple variables that can be assigned, namely lr, beta_1, beta_2, epsilon, decay and amsgrad.

Lr is short for learning rate. This value is α in the previously stated learning function.

$$A_{n+1} = A_n - \alpha \cdot (J_{(A)}')$$

In which A_n is the old position, A_{n+1} the new position, α the learning rate and $J'_{(A)}$ the derivative of the cost function.

We have set the learning rate to 0,001. This is also the default value in Keras and Tensorflow models.

The values beta_1 and beta_2 are the so called 'decay rates' of the learning. They along with lr they define the final learning rate. This final learning rate α_t is defined by the following function[48].

$$\alpha_t = \frac{\alpha \cdot \sqrt{(1 - \beta^t{}_2)}}{(1 - \beta^t{}_1)}$$

This learning rate α_t is subsequently used for calculating the step towards the local minimum.

The epsilon is a value "that is used for numerical stability"[49]. The value 'None' does not mean it is not present, or zero, it merely means that Epsilon is set to its default value of 10^{-8} [50]. This is the optimal value for the epsilon, as stated by Adam's inventors Kingma and Ba[51].

[48] Kingma, D., & Lei Ba, J. (2015). Adam: a method for stochastic optimization. Retrieved on 5 december 2018, from https://arxiv.org/pdf/1412.6980.pdf

[49] Tensorflow. (2018, 20 november). tf.train.AdamOptimizer | Tensorflow. Retrieved on 5 december 2018, from https://www.Tensorflow.org/api_docs/python/tf/train/AdamOptimizer

The decay variable is set to 0.0, as is the default for the adam optimizer. This value is the "decay of the learning rate over each update"[52]. Learning rate decay was not mentioned in the original paper of Kingma and Ba, and we have stuck with Keras' default because it seemed to influence the network negatively.

The last variable is AMSGrad. If this variable is set to 'True' the Adam optimizer will convert into a different version of it, called the AMSGrad optimizer, created by S. Reddi, S. Kale and S. Kumar. The difference between Adam and AMSGrad is that AMSGrad aims to achieve better results by having the learning rate influenced by "long term memory of past gradients"[53]. This can be seen as an extension of the Adam optimizer. We have decided not to use the AMSGrad optimizer, as it did not provide the same amount of adaptability the regular Adam optimizer provided, in addition to not showing any notable difference in the training process.

The last line of code calls the function model.compile(loss_calculator, optimizer).

The loss_calculator is different depending on the input, which will be defined later in the code. This is a function that calculates the loss of the prediction, following the principle explained in chapter 5, Artificial Intelligence. The optimizer is the variable adam, which has been defined in the line of code above.

The following piece of code is the training and testing of the network.

```
    model.fit(x_train, y_train, epochs=200, batch_size=5,
validation_data=(x_test, y_test))
    score = model.evaluate(x_test, y_test, batch_size=1)
    print(score)
```

The Keras command model.fit(dataX, dataY, epochs, batch_size) is the command for training the network on datasets X and Y, in our case our later defined variables x_train and y_train.

The value of epochs is the number of times the training data is forwardpropagated through the network. The batch_size is, as its name implies, the size of the batch before the backpropagation occurs, in this model 5.

The number of epochs has been defined as 200. Less epochs can lead to uncertainty in predictions and larger values make training the program take very long, thus it was set at this

---

[50] Keras. (n.d.). Optimizers - Keras Documentation. Retrieved on 5 december 2018, from https://Keras.io/optimizers/

[51] Kingma, D., & Lei Ba, J. (2015). Adam: a method for stochastic optimization. Retrieved on 5 december 2018, from https://arxiv.org/pdf/1412.6980.pdf

[52] Keras. (n.d.). Optimizers - Keras Documentation. Retrieved on 5 december 2018, from https://Keras.io/optimizers/

[53] Reddi, S., Kale, S., & Kumar, S. (2018). On the convergence of Adam and beyond. Retrieved on 5 december 2018, from https://openreview.net/pdf?id=ryQu7f-RZ

value. The batch size has been set to 5, as this seemed to be the optimal value. Larger or smaller batch sizes lead to higher overall loss and slower descent.

The variable validation_data is the data used to test the network between epochs using different data than that on which the network is training. The program will then return the loss and val_loss when the testdata is run through the network. These values give an indication of the progress of the training. These values stagnate after a certain amount of epochs. Using validation_data gives insight in when this occurs and allows us to find the ideal batch size and number of epochs.

The function model.evaluate is used to compute the loss value of testdata that is run through the model in test mode. This data is also used in the next bit of code, shown below.

```
x = 0
for x_indi in x_individual:
    x += 1
    print(x)
    indiArray = [x_indi]
    print(model.predict(np.array(indiArray)))
```

This bit of code has the completed network predict the speaker of a given test input. The value x = 0 will increase after every prediction, thereby keeping track of the predictions.

A loop is then called for the length of the imported value x_individual in which an array is made of x_indi (which is testdata that has not been run through the model before) on which the model will predict the outcome, whether it is Steven or Matthijs, by way of the function model.predict.

For the next bit of code we will deviate from the order the program was written in for better comprehension of the next part of our explanation.

```
f = open(trainFile + '.pckl', 'rb')
x_train, y_train, x_test, y_test = pickle.load(f)
f.close()

f = open('store_1_multi_test_2.pckl', 'rb')
x_individual, x_indi_test = pickle.load(f)
f.close()
```

This bit of code is in place to open our previously stored files form the data conversion, which we stored using pickle. The variables x_train, y_train, x_test and y_test which are used in the network are taken from the .pckl files from the conversion. The name “store_1_multi_test_2” is the name of the .pckl file we want to use for the network. This can be the name of any file we want, in this instance we have used this file.

The last bits of code are as follows.

```
typeMulti = True
if typeMulti:
    trainFile = 'store_1_multi_5_2'
else:
    trainFile = 'store_1_sigmoid_1'


if typeMulti:
    tensorModel(2, 'softmax', 'mean_squared_error', trainFile)
else:
    tensorModel(1, 'sigmoid', 'binary_crossentropy', trainFile)
```

First a variable called typeMulti is defined. In this case it is stated as 'True'. In chapter 5, Artificial Intelligence different activation functions have been presented and explained. The entire model uses the ReLU activation, except for the output layer. This layer had its activation function defined when the function is called. The reasoning behind this is because of the way the model is given form.

When having two possible outputs, there are two possible ways the output layer can be given form. A single neuron can be used, which means a sigmoid activation function is the most logical option to use in a classifier. However, another possible configuration is the use of two output neurons, which hold the possible outputs [0, 1] and [1, 0]. In this case the most logical activation function for the classifier would be to use a softmax function.

The data conversion program shown earlier in this paragraph has two possible settings for labeling the data, a sigmoid version and a multi version. Depending on the way the network is configured it requires differently labeled data. This data is also stored in files with different names.

If the variable typeMulti is 'True' a 'multi' file will be imported as training data. If it is false, a sigmoid file will be imported. The same idea holds true for the calling of the network itself. If typeMulti is 'True' the network will have 2 output neurons, a softmax activation function and it will use 'mean_squared_error' for computing the loss. If typeMulti is 'False' there will be a single output neuron which uses sigmoid as its activation function and 'binary_crossentropy' as its loss calculator.

The reasoning behind the split between multi input and binary input is important to understand. Sigmoid is great for classifying between two categories. Since there were only two speakers to be verified, sigmoid was a natural choice to use. However, we decided to experiment with using a multi-type input for the classification system with the idea in mind that the program needed to be able to expand. A sigmoid classifier is unable to differentiate between more than two classes. However, softplus is able to calculate the output for multiple classes. These inputs can be labeled as a list, in which speaker one has the value [1, 0, 0], speaker two has the value [0, 1, 0] and speaker three has value [0, 0, 1]. This way the database

can incorporate additional speakers without having to completely remodel the network. The only difference is an extra neuron that has to be added to the output layer.

7.4 Testing the neural network

7.4.1 Training data size

Training a neural network can not be done with a few sets of input data. Neural networks trained to recognize hand written digits make use of a library containing 60.000 samples used for training and 10.000 samples for testing the network[54]. Neural networks require a large amount of data to train. Using large data sets to train a network will help the network classify similar, though unidentical data because as it has trained on a more varied dataset.

Creating such a large dataset for speaker recognition is nearly impossible, as each sample is 3 seconds in length. A dataset of 1000 samples would require 50 minutes of continuous speaking, disregarding the margin before and after the recording and storing the samples. Recording 100 samples took about 30 minutes, six times the recording time. Moreover, the Fourier transform took about 3 seconds per sample, so that would take another 50 minutes of computation time. In total this means creating a 1000 sample database would take almost six hours without taking time to rest the voice.

Recording audio samples using the mobile application was faster, though of lesser quality. Labeling the recording was a slowing factor, which the app got rid of completely. This way recording, converting and predicting an audio sample could be done in about 6 seconds.

Most of our research was done using a database of 200 samples, made up in equal parts Steven and Matthijs. Testing was done in two different ways. First there was a small dataset of 15 samples using which the val_loss was computed. In addition to this the app was used to test the network with entirely unique data each time. Using the app we recorded 200 test samples on which the network achieved an accuracy of 99 percent, suggesting 100 samples per person are sufficient for training the network.

Even so, a larger database per person would likely be required when more speakers record training- and testdata. Humans can differentiate between two voices easily after having heard them a few times, but distinguishing 40 different people will take more time. The same holds true for artificial neural networks. As the number of possible outputs increases, more training data is required to effectively classify them.

[54] Brownlee, J. (2017, 18 october). Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras. Retrieved on 5 december 2018, from https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-Keras/

7.4.2 Converting the data

The first version of the neural network had a different way of using the Fourier graph as input. In version 2 of the network the Fourier graph was used as input by dividing it into 50 equal parts, where in each part the average y-value was used as input. This way the entire Fourier graph is 'compressed' into 50 pieces of data. By doing this all information was used as input.

Version 1 used the Fourier graph as input in a less complete way. Instead of compressing the graph we took coordinates on the graph as input. This meant 501 coordinates, equal distance apart along the x-axis, were put in an array. The position in the array represented the number of steps taken along the x-axis from the origin, the value in the array was the y-value of the coordinate.

Version 1 had an accuracy of 99% using this technique, while training lasted 40 epochs. Despite this number looking very high, the network was not always very certain. About once every 15 predictions the certainty was only 6/4. Version 2 also had an accuracy of 99% in addition to having a high certainty, with a 200 epoch training duration. The most significant improvement was the difference in the ability to recognize speakers based on the sentence "hey Google". Network 1 had an accuracy of 79% when performing predictions on these inputs, whereas network 2 achieved an accuracy close to 100%.

Moreover, the second network was more stable in training. The val_loss in version one fluctuated substantially. Each time the network was trained it had a different value, sometimes very low, other times very high. The val_los of version 2 consistently ended up at 0,3333. The certainty of this model was high as well. In the most extreme cases the uncertainty was 19/1.

Even so, both networks achieved over 90% accuracy on 'EEE' samples, which was the initial goal we had set for this research. The reasoning behind the decision to create a second network was to improve training time and try to improve the 'hey Google' score, both goals were achieved.

7.4.3 Layers in network 1

The first network had three hidden layers of 501 neurons. During testing it became clear that this was the 'ideal' amount of layers. Two hidden layers usually had between 80-95% certainty, with drops to 70 or even 60%, whereas a three hidden layer neural network consistently had a 99.99% certainty, with fewer drops toward the 70% range. A network containing four hidden layers seemed to improve the network, as the val_loss was lower on average. However, there was no significant improvement in the predictions the network made on testdata. As smaller networks are less computationally heavy, the total amount of hidden layers was kept at three.

The number of neurons in the input layer had been set to 501, a rather large number for an input layer. We chose to make it this large in order to encapsulate the entire Fourier graph. The number of neurons in the hidden layers was also 501. This was the configuration during most of our early testing. Smaller hidden layers compared to the input layer worsened the predictions.

Larger hidden layers slowed down the training process and did not yield any noticeable improvement of the network. Consequently, the size of the hidden layers was kept at 501 for most of the testing.

The performance of this network can be seen in Attachments section A.

7.4.4 Layers in network 2

The second network had three hidden layers of 50 neurons. Because the Fourier transform data was much more condensed a smaller network could be used for predictions. On the other hand, this network took 5 times as many epochs to train properly, taking about double the training time of the first network. During testing we found that hidden layers of 40 neurons proved to be equally as effective after training. Testing on this network was performed after a training time of 120 epochs, when convergence had already occurred (as can be seen in Attachments B).

The number of hidden layers in this network was kept the same as the first network, as a different number of hidden layers negatively impacted the network. The same holds true for the dropout configuration. As no improvements were achieved changing these values, they stayed the same for the second network.

The performance of this network can be seen in Attachments section B.

7.4.5 Layers in network 2 with three speakers

Initially the network was remained configured the same way, the only difference being the addition of an extra output neuron for the new speaker. During testing it became clear that the network performed less on inputs which were not trained on, such as very low pitched ‘EEE’ sounds, or ‘hey Google’. We wanted to see if increasing the amount of inputs would improve the performance. What we noticed was a more stable val_loss. Note that this val_loss was computed on 56 test samples instead of 15.

The number of hidden layers stayed the same for this network compared to the two-speaker configuration.

The performance of this network can be seen in Attachments sections C and D.

7.5 Performance of the networks

There are a few values of paramount importance when looking at the performance of a neural network, the loss and the val_loss. These two terms have been used in this paper before and explained briefly. The loss is the cost, or the amount of error, the network has during training. This value is used to compute how much the weights and biases need to be appended. The

val_loss is similar to the loss, but this value is not used during the improving of the neural network, as it is the cost the network achieves on the test data after each epoch.

Naturally, both the loss and val_loss should be as low as possible, as this means there is little error. Usually the loss will end up lower compared to the val_loss, as it is the value the network is actively trying to decrease. Therefore, when overfitting occurs the loss will be very low and the val_loss very high.

We have plotted the loss and val_loss of network 1 and network 2 in the Attachments sections A and B respectively.

When looking at the graphs in attachments A and B a few things are seen immediately. The loss usually reaches zero after around 20 epochs. The val_loss usually stagnates at around 0,3333. However, this does not mean one third of the predictions are wrong, as all of these configurations (excluding the configuration in Attachment A 4) had all test data predicted correctly, and with high certainty (above 99%). It should be noted that these networks only had 15 test data samples to work with. The value 0,3333 seemed to be the minimum a well configured network could reach. When training a network for only 10 epochs the val_loss could be lower, but the predictions would be very uncertain.

As stated before, both networks performed very well on ‘EEE’ test samples, achieving 99% accuracy over 200 training samples. This testing was done via the mobile application. However, the second network performed much better on training samples of ‘Hey Google’, where the first network achieved only 79% accuracy, the second network did not predict a single sample wrong.

We later added a third speaker into the database, using the second network with an extra output neuron for predicting on the data.

During part of the testing we increased the amount of input neurons from 50 to 100 for the three speaker network, meaning the Fourier graphs were translated more specifically. The predictions improved slightly in their certainty with about 0,1%. The greater difference however was the increase in training stability. Whereas the 50 neuron network had large fluctuations in the loss and val_loss, the 100 neuron network was extremely stable after 50 epochs, as can be seen in Attachments C and D.

When a third speaker was added to the second neural network there was a significant drop in performance. The accuracy of the network when predicting on ‘EEE’ dropped from 99% to 90%. The and the predictions on ‘hey Google’ from 99% to 76%. Strangely enough the network did not predict wrong on each speaker equally.

	EEE	Hey Google
Matthijs	100%	99%
Steven	90%	38%
Henriet	80%	92%
Average	90%	76%

Matthijs was predicted correct almost every time. Steven's 'EEE' samples achieved an accuracy of 90%, whereas his 'hey Google' samples were only predicted with 38% accuracy. This was to be expected, as the network did not specifically train on these samples. Still, the fact that only Steven's samples of 'hey Google' were predicted falsely was rather strange, as we expected a more uniform drop in accuracy across all speakers.

The cause of the large drop in the total accuracy of the network when predicting on 'EEE' samples was largely caused by the Henriet. On average the network achieved an 80% accuracy when predicting on Henriet's 'EEE' samples. However, there were drops toward the 47% range and peaks at 100%, depending on what local minimum was reached. Each time the model is trained, the convergence occurs at a different local minimum, which seemed to impact the Henriet training data rather substantially compared to that of Steven and Matthijs.

This is a great example of the impact local minima have in the accuracy of neural networks. Sometimes the network achieved 95% total accuracy on test samples and other times only 80%. This also raises the question on how dependable the percentages on test samples actually are. If the performance of a network can fluctuate so much on the local minimum it happens to reach, how trustworthy is the network actually?

The reason Henriet above everyone was impacted so heavily by this could be traced back to the Fourier transform graphs. When looking in section E of the Attachments it can be clearly seen that Henriet has the least peaks in her graphs. There are less distinguishable features in her Fourier graphs compared to Steven and Matthijs, possibly leading to them being 'overtaken'.

What also might have impacted the drop in accuracy was the use of a different microphone during the recording of Henriet's train and test data. The training data of Steven and Matthijs was recorded using a microphone connected to a PC, whereas Henriet's training data was recorded using the same microphone connected to a phone. The differences in the way the audio was processed could have influenced the network, as the training samples were not all recorded in a uniform way.

It should be noted that this testing was done on the 50 division network, not the 100 division one. As the 100 division can define signatures better, the later parts of the Fourier graphs would have been encapsulated better, perhaps leading to better performance. However, we were not able to test this due to time constraints.

7.6 Summary

The research we have conducted was divided in three main parts. The recording of the voices, which was the easiest part of the research. This does not mean it was the least important part, as it is the foundation of all our research. To provide the network with the best input possible the way voices were recorded was standardized for to uniform the recordings.

The Fourier transform code was the next part of the research. Raw .wav audio files were converted to data that could be used as input for the network. Properly ordening the transformed audio was imperative for it to be used as input. The transformed data had to be labeled properly depending on the type and the file it was stored in had to be named accordingly.

When programming the networks it was important the code was clear and easily edited. Using Keras proved to be a good choice as it helped keep the code clean and being able to change the code quickly. The trickiest part of using our neural network was tweaking all the features of the network so that it achieved optimal results with the least amount of training required. Number of layers, neurons, dropout, optimizers and loss calculators were all variables that had to be manually adjusted.

After hours of testing and tweaking the program the final network was able to achieve 99% accuracy in predicting voices between two people, an astonishing percentage. The change that was made in the way the Fourier transform was used as input increased the network accuracy on the input “hey Google” drastically. However, the addition of a third speaker caused a considerable drop in accuracy. Performance on ‘hey Google’ dropped the most, to 76%, and on ‘EEE’ the accuracy dropped to an average of 90%.

8. Conclusion

The thesis question for our research was the following:

"**Is it possible to create a deep neural network that can recognize a voice with an accuracy above 90 percent?**"

Additionally, we formulated two partial questions:

1. Will the accuracy of the predictions remain the same when a third speaker is added?

2. How does a decreased number of neurons affect the accuracy of the predictions?

The research we have conducted was centered around the main thesis question. The first partial question is an inherent part of this research, whereas the second partial question was focused on the ability to combine different technologies.

During our research we created a neural network that can classify two, and later three, people based on only their voice. The Fourier transform was used for transforming the audio samples, after which it was used as input for the network. After training, this network had a 99% accuracy in classifying between two speakers, meaning our hypothesis was correct.

When a third speaker was added to the network the performance dropped substantially. Predictions on 'hey Google' to 76% and on 'EEE' to 90%. Interestingly, Matthijs was predicted correctly 99% of the time on 'EEE' samples, whereas Steven and Henriet were predicted correctly 90% and 80% of the time respectively.

Henriet's performance was the most curious. Sometimes the network achieved 100% and other times merely 47% accuracy. The local minimum the network achieved impacted the accuracy of the predictions on her test data very much. Why this was especially her data can be traced back to the recording of the training data, which was recorded using the same microphone, but through a different device. Henriet's data was added to the training database using the app, meaning the audio was processed by the iPhone SE she used. The quality of these recordings is naturally different compared to the PC by which Steven and Matthijs' audio was processed. Additionally, the Fourier graphs of Henriet have less distinct peaks compared to both Steven and Matthijs. Henriet usually had 4 peaks, whereas Steven and Matthijs usually had additional peaks further right in the graph.

The first network took 40 epochs to train, which it did on a database of 100 samples per class, meaning 200 samples in total. After training this network could differentiate these speakers with an accuracy of 99 percent. This accuracy was measured using the app, where we both gave 100 new inputs. These inputs were of the same type the network had trained on, three continuous seconds of "EEE".

Furthermore, the second network was also able to identify speakers when the input was unlike the training input. We both used the app to give 60 inputs to the network, where we said: "Hey

Google". Despite the network never having trained on these inputs it still achieved a 79 percent accuracy.

The second network took 200 epochs to train. While this may seem like a step back, the network was ten times smaller than the first network. Consequently, training time was significantly decreased. Even using only 40 epochs this network trained much faster than the first network, while having the same certainty in its predictions.

After training 200 epochs this network also achieved a 99% accuracy in classifying "EEE" input. However, the program had an astonishing improvement in classifying based on the "hey Google" input, a 100% accuracy. Of course, a 100% accuracy is impossible to attain as the testing is susceptible to chance. Nonetheless it is clear the second network improved upon the first in this regard. This could be in large part due to the fact the entire Fourier transform was used as input instead of just a few coordinates. In addition to this the training process of this network was more stable. The val_loss of the first network ended up at a random value each time the network was trained. On the other hand, the val_loss of the second network consistently ended up at 0,3333. The graphs of the loss and val_loss are shown in Attachments A through D. These graphs show the amount of epochs to reach convergence, as well as the unpredictability of the network (see the difference between Attachments C and D).

It is important to keep in mind the fact that an untrained network with a set amount of possible outputs will have a base accuracy. An untrained network will not have high certainties, instead it will give all outputs some, seemingly random, value.

During testing we let the first network predict outputs without having been trained beforehand. This yielded the expected results. Both classes got an equal 50/50 split in confidence, with variations in the single percentage range. This gave a natural 50% accuracy for the untrained network. Training this network obviously improved the accuracy rate, up to the previously stated 99%. This is a significant change, but a base accuracy of 50% makes it seem less of a development.

However, as the confidence of the predictions was so low (50/50), they should not be considered valid. For our testing we considered a prediction viable only if the certainty was 80% or higher, which it was 100% of the time, usually is was a 99,9% certainty.

When a third speaker was added the certainty of the predictions did not deteriorate much. The certainty of the predictions stayed in the 99% range. The untrained network with three output neurons had predictions of [0,3333 0,3333 0,3333], or a 0% certainty. The trained network achieved certainties of 99% and higher.

Moreover, when we shortened the length of the Fourier divisions so that the graph was divided in 100 parts instead of 50, the loss and val_loss became substantially more stable, as shown in Attachments C and D. Consequently, the predictions went up in certainty as well.

8.1 Possible improvements and further research

8.1.1 Increasing dependability of the network

When a third speaker was added to the network the impact of different local minima was shown. The performance of the network fluctuated between 80 and 95 percent. As the local minimum the network happens to reach influences results so substantially, one is left to wonder how dependable the network actually is. There is a 15% difference in network accuracy depending on the local minimum that is reached.

More research is required to find out how much the network is impacted and how to prevent this from happening in the future. We had very little time left to do research and perform tests on this subject. The initializing of the network was set as 'uniform' in our network. Different initializations change the direction the program will compile in. The dropout has a part in this as well, adding a randomness factor to the training. In the end, we had too little time left to properly go through on these ideas, but further research would certainly shed some light on how this aspect of the neural network can be improved.

8.1.2 Handling unrelated input

After training, both programs were very, for lack of a better word, absolute in making their decision. If a completely unrelated person would give input for the program the outcome would always be very certain. Instead of a more realistic 0% the output always was very determined. This phenomenon could be explained due to the fact the network only has two possible classes and two, rather distinct, types of input. If the network trains too much on these specific data types it can lead to a type of overfitting, where the network divides the data between the classes, and everything that falls even a little bit into either class will be confidently classified as Steven or Matthijs, there is no 'grey area'. The addition of Henriet seemed to decrease this absoluteness somewhat, though not very substantially.

There are a few possible ways this phenomenon could be prevented. First a classical approach to preventing overfitting could be used by reducing the amount of epochs trained or using dropout. This will however make the network less effective in classifying valid inputs as well. Another possibility is the addition of more possible output classes, so that such a division is less likely to be formed. With enough possible classes the chance of having a unrelated sample being placed in but one class will be very small. Finding balance between classifying valid inputs and not overfitting invalid inputs would require great amounts of testing.

8.1.3 Improving the training process

Currently the network requires a lot of training data before attaining a high accuracy during testing. Our results were achieved using a database of 200, and later 300, samples, that took over an hour to record. The network proved to be able to predict correctly when provided with

less training data, but the accuracy dropped significantly, even disregarding certainty and base accuracy. When more possible outputs are provided, this accuracy will drop even further.

Before voice recognition can be used in normal day-to-day environments the training process will have to be improved. Recording 100+ data samples is not attractive for consumers. Possibly a different way of representing the data to the network will lower the amount of required training samples. This means that something different than the Fourier transform will have to be used to transform the voices into input for the network.

8.1.4 Handling fluctuations in the voice

Another problem with our voice recognition program is the fact that an individual's voice is prone to change. Soar throats and fevers can cause significant differences in the sound of a voice. Moreover the network has difficulties when a voice is higher or lower pitched. Changing the pitch and way input was said influenced the accuracy of the network. As we used Fourier transform, which is specifically designed to compute the frequencies of a sound, changing the pitch (frequency) of a voice will therefore impact the Fourier transform, and by extension the input as a whole.

8.1.5 Recognizing voices using different audio inputs

The Fourier transform has proven to be an effective way to transform raw audio into usable input for a neural network, as a 99% accuracy rate was achieved with two speakers and 90% with three speakers. Nevertheless, the Fourier transform lacks in a few aspects. The input was very sensitive to changes in the voice pitch and inputs that are dissimilar to "EEE" gave results with low certainty, especially long sentences. The Fourier transform of long sentences is very different compared to a single "EEE" transformation. For the network to be able to classify people based on any given sentence another way of representing the speakers as a whole will have to be used, as the Fourier transform is not fit for such inputs.

8.1.6 Reconstructing voices

Now that we have been able to classify people based on their voice, a natural continuation would be to 'reverse' the process. Software such as google translate already has the ability to convert text to speech. It could be possible to reverse the voice recognition process to get a 'model' of someone's voice. By conjoining this with text-to-speech software a program could be created that is able to construct audio this sentence as if it was said by the test subject. The sentence would be said using the test subject's voice. However, it would require a more complete comprehension of the voice than a simple Fourier transform of "EEE" to create such a model. In short, before a program like this could be created, there will have to be found a way

to more completely encapsulate a human voice, that can also be used in conjunction with a text-to-speech converter.

8.2 Summary

Our thesis is "**Is it possible to create a deep neural network that can recognize a voice with an accuracy above 90 percent?**". The answer to this question is: "Yes we are able to train a neural network to recognize a voice with 90 percent accuracy". On 'EEE' inputs the second neural network achieved 99% accuracy using only 200 training samples and 90% accuracy when using 300 training samples. In addition to this, the network only takes fifteen to twenty seconds to train properly. As was expected, on inputs that do not have clear "EEE" sounds in them the network proved to be significantly worse in making predictions. Additionally, the amount of data samples it took to train properly was very large, though comparatively small for neural networks in general.

The addition of the third speaker caused a considerable drop in accuracy, though it stayed above 90%. These drops were caused by the local minimum the network happened to reach. Depenidng on how the model was compiled the performance could be above or below 90%, though on average it achieved 90%.

Even so, the thesis question can be answered positively. Possibilities for improvement of the current network and suggestions for topics that require further research are namely the way the voice is represented as data. The Fourier transform is effective but flawed. The network itself is effective in its designated task, but it requires a lot of training data. With further research these flaws will be improved and the technology of voice recognition shall be one step closer to perfection.

9. Applications

9.1 Verification and identification

9.1.1 Security

Could there be applications for voice recognition after it has been researched and developed further? We have come up with a few ideas. The first application that comes to mind with an identification program is security. Voice identification could be used to unlock devices or apps in much the same way a fingerprint or face scan works in modern cell phones. The danger is in this case that our program is specifically designed to recognize recorded voices. A voice recording could be used by someone with bad intensions to get access to sensitive information.

To prevent this voice recognition could be used in conjunction with speech recognition, a technology already widely used around the world. Voice assistants such as Siri and Alexa use speech recognition since 2011. By combining voice identification with speech recognition a far more secure program can be created.

For example, to unlock a phone a series of three words could be shown on screen, like “cranberry, elephant, Microsoft”. These would be randomly selected from a database containing thousands of possible words. A speech recognition program would verify is these words have been said, after which a speaker recognition program will verify if the speaker is in fact the right person. This way a previous recording of a person’s voice would not be sufficient to fool the program, as the words said would prevent this from happening.

To further increase security a small bit of software could keep track of combinations that have already been said, so that a unique combination of words would be said each time. This way, a 1000 word database can provide close to one billion different word combinations, and a 10.000 word database close to a trillion different combinations. If the voice identification software is sufficiently developed the only way to fool the program would be to have a recording of all 10.000 possible words and playing the ones requested when unlocking.

9.1.2 Investigation

In addition to providing security voice identification could also be used in crime fighting. Audio recordings of crimes could be used to identify the criminal that is speaking. The audio could be run through a database the same way fingerprints are, and if a positive match is found this suspect could be investigated further. This way wearing a mask and gloves during a robbery would not be enough for criminals to remain anonymous, as even saying a single phrase would help police officers in identifying the culprit.

9.2 Voice assistants

Aside from security and criminal investigations voice recognition could also be a quality of life improvement. A voice assistant such as Siri or Alexa could identify the person speaking. A voice assistant could save preferences of people. For example, you could say to the assistant “play my favorite playlist”, leading to the assistant playing the specific speaker’s favorite playlist. These are small, arguably unnecessary improvements, but they could nonetheless be implemented in every voice assistant if there is demand.

9.3 Summary

In short voice recognition is software that has possible, but not necessary, applications. Speaker recognition can be used for security, especially in conjunction with speech recognition. The question is whether or not this is actually necessary with other ways for protection already in place. Fingerprint scan and facial recognition are widely used technologies that are secure and dependable. Speaker verification will have to compete on a market with a range of possibilities.

Speaker recognition can however improve voice assistants. Again, possible additions will make them easier and more accessible to use, though are not necessary by default. Voice recognition is a niche technology that has applications, but can be seen more as a luxury rather than a necessity. Still, before speaker recognition can be used in this way, more research should be done into improving accuracy. Our goal of over 90% accuracy was met, but when dependable security is required, especially with larger databases, accuracy will need to be improved.

10. Review

Early research, spring and summer 2018

Our research had its share of flaws. We started in April 2018, with very little knowledge of artificial intelligence and its aspects. We were overwhelmed by the sheer scope of different facets and possibilities in the field of AI research. We started by reading online on sites such as Wikipedia to get an idea on what subject within AI we would research. We spent many hours on the internet where we tried to learn as much of different subjects as possible.

These hours would have been better spent increasing the scope of our knowledge instead of trying to obtain in depth knowledge of each subject. In addition, our focus should have laid more on applications of AI rather than theoretical research. Moreover, we spent lots of time watching MIT lectures and looking on AI crash courses. The MIT lectures where too theoretical and the crash courses were too specifically focused on certain problems.

On the other hand the brainstorm sessions we had were very productive. During these sessions we eliminated subjects which were either impractical or beyond our capabilities. We also went around and tried to think of as many possible applications of AI that we could research.

Start of voice recognition, early autumn 2018

At the end of summer 2018 we decided on our final subject, voice recognition. After the subject had been chosen our productivity rose substantially. We could research on AI with a specific goal in mind. We visited physics teachers for advice on how to handle voices and we started specifically researching neural networks. We got the Fourier transform code working within a few days and already had an idea on how to input them in a neural network in spite of the fact we were not yet certain on how to create a neural net. Once we found out how to create neural networks using Keras we got the code working in a few days

In hindsight we should have started researching on Tensorflow and Keras sooner. These libraries made creating neural network much easier than we had expected. Tutorials on the internet created neural networks neuron by neuron, whereas in Keras they could be created in a few lines of code. Even so, this part of the research was very productive.

Stagnation in project, mid-autumn 2018

Even though the code was running without errors it did not seem to be working properly. Even though the loss was decreasing rapidly another value, the val_loss, fluctuated heavily. Every time the model was trained this value reached another, seemingly random, value. As the val_loss was the loss the model had on the testing data, we were under the impression that the model suffered from overfitting.

Weeks later we realized that this may not have been the case. We used the Keras function model.predict to see if the model was indeed bad at predicting on test data. We found, to our surprise, that the model predicted all testdata perfectly, and with high certainty.

Had we used the model.predict function earlier we would have spared over a week of research time. Our narrow vision, relying solely on the val_loss as indication of the accuracy, cost us valuable time.

Rapid progress, late autumn 2018

After we realized the network was working properly we started making great progress in our research. We started working on the paper and decided we would create a mobile application for easier use of the network. Over the span of six weeks we thoroughly tested the network and Keras' different capabilities and improved the architecture of the network. After the network had been optimally configured we recorded over 300 samples to test the network and gather percentages of accuracy.

In November we received a suggestion from a student of the Hague University who was following a machine learning course regarding the use of the Fourier transform as input. Instead of coordinates he suggested a more complete way to encapsulate the Fourier graph. During testing the results on the sentence 'Hey google' improved substantially, from 79% to 99%.

We were writing the paper during this time as well. The chapters were written over the span of a few weeks and without the need to revise large sections of the paper. All in all the research was progressing in great paces during this time.

Concept version, early December 2018

The end of November was reached very fast, and our focus shifted from testing the network to writing the paper. Days before the deadline the paper still was not finished. The conclusion and large parts of the chapter on the research were still missing. On the day of the deadline we were still writing, and when handing in the concept version we had not yet revised any of our recent writing.

The mistake we had made was underestimating the amount of work the paper would require to finish. We should have written more of the paper in the free time we had the weeks before the deadline, instead of the deadline day itself. Writing a paper is not merely writing text, but also deciding which part of the text need to be improved or omitted. Nevertheless the paper was complete and handed in on time. However, we should have learned from this mistake the first time, as we immediately made it again.

Stand still, December 2018 and early January 2019

During the Christmas holiday work on both the paper as well as the program stagnated. We did not do anything on the research despite the fact we had lots of free time, which would later

lead to lots of stress and hasty work in the last week before the final deadline. We should have spent this time more wisely and finished the research before the end of the holiday.

Final sprint, January 2019

After the Christmas break we realized how much still had to be done in only 2 weeks of time, including the addition of a third speaker, adding further functionality to the mobile application and a great amount of writing on the paper. We made a timetable using the scrum project management technique, where we divided the remaining time in sprints, over which we spread all the remaining work equally. Even so, there was much work and little time. Over these two weeks deadlines for normal school projects had to be met as well, increasing the stress even more. Even though the paper was finished on time, we realized we should have done more work during the Christmas break instead of working double-time during the semester itself.

In the end the paper was handed in before the deadline. Even though the process could have been smoother, and mistakes were made along the way. We feel the paper is complete, and we are proud of the final product.

11. Summary

11.1 English

Our goal was to create a program that can identify speakers with an accuracy over 90%. In this goal we have succeeded. The research consisted of two parts, converting audio files to input, and creating a program that is able to make predictions based on this input.

After the voices were recorded, we converted the audio files using the Fourier transformation. This gave us a graph of the frequencies the voice was made up of. With the same sound, this graph is unique for every person, and thus viable for use in identification. By dividing this graph into segments and taking the average of the graph for each segment we 'compressed' the graph into 50 values.

These 50 values were used as input for our program. To compute the data we decided to use a neural network, a type of AI which is based on the human brain. Neural networks are structured in layers, where the data is given to the input layer and computed in the hidden layers, after which the result will be presented in the output layer.

For our network we created an input layer of 50 neurons, one for each piece of the Fourier graph. The data was then forwarded through three hidden layers, each containing 50 neurons. The data was the presented in the output layer, which consists of three neurons, one for each of the three speakers.

Using these techniques, the network managed to achieve an accuracy of 99% when tested on two hundred new samples with two speakers, which was well over our initial goal of 90%. Thus, by using deep learning algorithms we have created a program that can identify known speakers with an accuracy over 90%.

11.2 Dutch

Het was al vrij snel duidelijk dat wij onderzoek wilden gaan doen naar artificiële intelligentie. Dit was echter een erg breed thema maar na veel brainstormen kwamen we toch op een specifiek vraagstuk uit. Stemherkenning werd het uiteindelijke onderwerp. Ons doel was gezet op het maken van een programma dat sprekers kon herkennen aan de stem met een minimale nauwkeurigheid van 90%.

Ons onderzoek bestond uit 3 onderdelen, het opnemen van de stemmen, het verwerken van de stemmen en het maken van een programma dat iets met deze input kon. Door de sectie natuurkunde werden wij in de richting van de Fourier transformatie gewezen, een wiskundige bewerking die frequenties uit een stem kan halen. Deze frequenties vormen bij elk persoon, gegeven dat er gebruikt wordt gemaakt van hetzelfde geluid, een unieke grafiek, die wij kunnen gebruiken om mensen te herkennen.

Er zijn verschillende soorten AI met verschillende manieren waarop ze functioneren. Artificiële neurale netwerken zijn een van de meest gebruikte types AI tegenwoordig, en hun kenmerken waren geschikt voor ons onderzoek. De Fourier grafiek deelden wij op in 50 gelijke delen, waarvan wij de gemiddelde y-waarde namen. Dit gebruikten wij als input voor het neurale netwerk.

Een neuraal netwerk heeft een ingangslaag, waar de data ingegeven wordt, een of meerdere verborgen lagen, waar de berekeningen plaatsvinden, en een uitgangslaag, waar het uiteindelijke resultaat wordt gegeven.

Wij hebben enkele honderden opnames gemaakt van onze stem, elk drie seconden met een continu ‘EEE’ klank. Na het verwerken van deze opnames worden deze tientallen keren door het programma gebruikt om te trainen, waardoor de verbindingen in de verborgen lagen geoptimaliseerd worden.

Het netwerk dat wij hadden gemaakt bestond uit een ingangslaag van 50 neuronen, één voor elke inputwaarde, drie verborgen lagen van elk ook 50 neuronen en een uitgangslaag van 2 neuronen.

De uiteindelijke resultaten die het netwerk behaalde overtroffen onze verwachtingen. Het netwerk had met twee personen een accuraatheid van 99%, meer dan ons doel van 90%. Wij hebben dus door gebruik van de Fourier transformatie en een neuraal netwerk een programma weten te maken dat sprekers kan herkennen op basis van een 3 seconden lange audio opname met een accuraatheid van 99%.

12. Logbook

Hours Matthijs Ates

Date	Time	Activity
17/4/2018	1 hour	Reading about AI
18/4/2018	1 hour	Reading about AI
20/4/2018	20 minutes	Reading about AI
28/4/2018	30 minutes	Reading about AI
06/5/2018	45 minutes	Lecture MIT
10/5/2018	50 minutes	Lecture MIT
13/5/2018	50 minutes	Lecture MIT
14/5/2018	50 minutes	Lecture MIT
15/5/2018	1 hour + 40 minutes	Lecture MIT
24/5/2018	45 minutes	Lecture MIT
31/5/2018	4 hour + 45 minutes	Sources lecture, looking for sources, brainstorming and defining our timeframe
01/6/2018	1 hour	Brainstorming and choosing thesis
18/8/2018	2 hours + 30 minutes	Videos: Essence of Linear Algebra
27/8/2018	2 hours	Seeking out learning material and expanding framework paper
10/9/2018	1 hour + 20 minutes	Working out second thesis, spoke to physics teacher regarding voice recognition and partial questions
12/9/2018	5 hours	Working out Fourier transform in python, defining thesis, video on Fourier transform
13/9/2018	6 hours	Worked out Fourier transform using testdata
16/9/2018	1 hour + 5 minutes	Videos on neural networks
20/9/2018	1 hour + 45 minutes	Tutorial, building a neural network in python
21/9/2018	3 hours + 20 minutes	Learning about creating a neural network. Creating and training a neural network, evaluating results
23/9/2018	1 hour	Testing neural network by computing values and differentiating functions
21/10/2018	1 hour + 30 minutes	Finding information on Tensorflow and defining tems for clarity
22/10/2018	2 hours	Reading guides regarding neural networks in Keras and activation functions
23/10/2018	3 hours + 45 minutes	Finalizing, testing and optimalizing network, recording audio samples
24/10/2018	3 hours + 45 minutes	Downloading code from github, testing network, recording audio samples
25/10/2018	2 hours	Converging Keras and conversion files, recording audio samples, testing code

04/11/2018	1 hour	Starting paper chapter AI
06/11/2018	1 hour + 20 minutes	Reading papers on speaker recognition
07/11/2018	20 minutes	Continuing paper chapter AI
08/11/2018	1 hour + 30 minutes	PWS meeting supervisor, continuing paper chapter AI & machine learning
11/11/2018	45 minutes	Paper section voice_rec & NN's and neural networks
12/11/2018	2 hours + 20 minutes	Paper section neural networks
16/11/2018	2 hours + 15 minutes	Paper section neural networks and start voices in general
17/11/2018	50 minutes	Paper chapter voices in general
18/11/2018	1 hour + 35 minutes	Paper section Fourier transform
19/11/2018	1 hour + 15 minutes	Paper chapter Fourier transform and inserting pictures NN
22/11/2018	45 minutes	Paper section recordingvoices
28/11/2018	40 minutes	Paper section recording voices & transforming voices
05/12/2018	5 hours	Paper section neural networks & creating the network
7/12/2018	4 hours + 50 minutes	Paper section using the NN, testing the network and finishing chapters 1 and 2
8/12/2018	35 minutes	Paper section applications
9/12/2018	4 hours + 25 minutes	Paper conclusion and abstract, doing testing on untrained network, testing network 2, listing sources
10/12/2018	2 hours + 20 minutes	Finishing using the neural net, making to do list, finishing paper, writing summaries, front page, etc.
8/01/2019	30 minutes	Making final planning + working out app improvement ideas
9/01/2019	1 hour	Technical design app
12/01/2019	30 minutes	Writing summary (English)
13/01/2019	2 hours	Adding labels to pictures, improving writing, writing functional design
15/01/2019	4 hours	Testing and improving new app, writing summary (Dutch), writing first parts of review, gathering data
16/01/2019	5 hours + 40 minutes	Working on and testing of application and network, recording samples, writing on chapter research, gathering data, fixing onedrive file
17/01/2019	5 hours + 10 minutes	Gathering data + Fourier graphs, writing on chapter research, finishing paper in general
18/01/2019	50 minutes	Writing on philosophy
19/01/2019	1 hour + 10 minutes	Incorporating feedback, finishing paper
20/01/2019	5 hours + 20 minutes	Incorporating feedback, finishing paper, conclusion
21/01/2019	1 hour + 50 minutes	Finishing paper

Total: 110 hours + 55 minutes

Urenlijst Steven van den Wildenberg

Date	Time	Activity
18/4/2018	1 hour	Reading
30/4/2018	2 hours + 5 minutes	Brainstorming & search for information
13/5/2018	45 minutes	Lecture MIT
29/5/2018	1 hour + 30 minutes	Reading
31/5/2018	4 hours + 45 minutes	Sources lecture + finding sources + brainstorming + making a planning
01/6/2018	1 hour	Brainstorming and choosing thesis question
22/6/2018	50 minutes	Reading
24/7/2018	1 hour	Setup idea & podcast think tank about AI
27/7/2018	2 hours	Search for learn material and expand research idea
28/7/2018	1 hour + 30 minutes	Part of the video's: Essence of Linear Algebra
29/7/2018	1 hour	Rest of the video's: Essence of Linear Algebra
02/8/2018	2 hours + 10 minutes	MIT Linear Algebra lectures 1-4 on 1.5x speed
03/8/2018	1 hour + 40 minutes	MIT Linear Algebra lectures 5-7 on 1.5x speed
04/8/2018	2 hours + 10 minutes	MIT Linear Algebra lectures 8-11 on 1.5x speed
14/8/2018	2 hours + 30 minutes	Book: Introduction to linear algebra fourth edition, chapter 1 and a part of chapter 2
16/8/2018	1 hour + 30 minutes	Excercises 1.2 and 1.3
10/9/2018	1 hour + 20 minutes	Figure out thesis statement + ask physics teacher for more information
12/9/2018	5 hours	Fourier transformation/ python/ thesis/ research requirements
13/9/2018	6 hours	Worked out Fourier transformation with testdata, pitch and beginning of paper
19/9/2018	2 hours	Research STFT and search for better replacement of Fourier transformation
20/9/2018	1 hour + 40 minutes	MFCC plotting in python and search for documentation/info alternatives
21/9/2018	1 hour + 20 minutes	Learn about making a neural network together with Tensorflow
28/9/2018	1 hour + 20 minutes	Write introduction
21/10/2018	1 hour + 30 min	Research Tensorflow + define complicated terms

22/10/2018	4 hours	Clarity Keras, get y-values Fourier transformation in python and compare graphs of vowels
23/10/2018	5 hours + 30 minutes	Create train data, convert train data and test out the train data on the neural network
24/10/2018	4 hours	Record train data and remove some inconsistent train data
05/11/2018	2 hours	Search and test Tensorflow speaker recognition on the internet
08/11/2018	30 minutes	PWS conversation with mentor
12/11/2018	1 hour + 20 minutes	Work on neural net
19/11/2018	3 hours	Make test version MFCC neural net
28/11/2018	6 hours	Make app + backend app
29/11/2018	4 hours	Finish backend app
07/12/2018	3 hours	Editing/improving and writing new text + add new code
10/12/2018	3 hours	Describing Fourier conversion code, editing paper in general
11/01/2019	3 hours	'Re-creating' the iOS and creating the android application
12/01/2019	5 hours	Continue with creating the application
13/01/2019	10 hours	Continue with creating the application (fixing bugs mostly)
14/01/2019	5 hours	Finishing the android and iOS application
15/01/2019	1 hour	Working on and testing of application
16/01/2019	3 hours	Fixing third speaker and debugging the application
17/01/2019	3 hours + 30 minutes	Editing paper and added Python explanation
20/01/2019	4 hours + 30 minutes	Editing last details of the paper
21/01/2019	2 hours	Finishing up the paper

Totaal: 116 hours + 35 minutes

13. References

3Blue1Brown. (2017, 16 october). Gradient descent, how neural networks learn | Deep learning, chapter 2 [Video]. Retrieved on 12 november 2018, from https://www.youtube.com/watch?v=IHZwWFHWa-w

3Blue1Brown. (2018, 26 january). But what is the Fourier Transform? A visual introduction. [Video]. Retrieved on 18 november 2018, from https://www.youtube.com/watch?v=spUNpyF58BY

Baras, J., & LaVigna, A. (1990, december). Convergence of a neural network classifier. Retrieved on 9 december 2018, from https://pdfs.semanticscholar.org/b263/9046c7d99e99119a2569fe58cedd75b000d6.pdf?_ga=2.7215792.38168314.1544376574-1239280478.1544376574

Bijterbosch, J., & from Wijk, P. (2015). *Nectar 3e editie biologie 5 VWO leerboek* (3e ed.). Groningen/Houten, Netherlands: Noordhoff.

Brownlee, J. (2016, 22 september). Supervised and Unsupervised Machine Learning Algorithms. Retrieved on 8 november 2018, from https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/

Brownlee, J. (2017, 30 march). Dropout Regularization in Deep Learning Models With Keras. Retrieved on 5 december 2018, from https://machinelearningmastery.com/dropout-regularization-deep-learning-models-Keras/

Brownlee, J. (2017, 18 october). Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras. Retrieved on 5 december 2018, from https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-Keras/

Brownlee, J. (2018, 27 april). A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size. Retrieved on 16 november 2018, from https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Clarke, J. (2018, 9 march). Can Artificial Intelligence Help With Depression? Retrieved on 20 january 2019, from https://www.verywellmind.com/can-artificial-intelligence-help-depression-4158330 (image on title page)

Domingos, P. (2012, 1 october). A Few Useful Things to Know about Machine Learning [Scholarly article]. Retrieved on 8 november 2018, from https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf

Fagella, D. (2018, 29 october). What is Machine Learning? Retrieved on 8 november 2018, from https://www.techemergence.com/what-is-machine-learning/

Google. (2018, 1 october). Reducing Loss: Gradient Descent | Machine Learning Crash Course | Google Developers. Retrieved on 12 november 2018, from https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent

Google. (2018, 1 october). Reducing Loss: Learning Rate | Machine Learning Crash Course | Google Developers. Retrieved on 12 november 2018, from https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate

Imperial College London. (n.d.). Neural Networks. Retrieved on 11 november 2018, from https://www.doc.ic.ac.uk/%7End/surprise_96/journal/vol4/cs11/report.html

Improve Shallow Neural Network Generalization and Avoid Overfitting - MATLAB & Simulink. (n.d.). Retrieved on 5 december 2018, from https://www.mathworks.com/help/deeplearning/ug/improve-neural-network-generalization-and-avoid-overfitting.html;jsessionid=fbb36bbaaea534e816b5b7bd4b93

Intuitive understanding of Euler's formula. (n.d.). Retrieved on 19 november 2018, from https://betterexplained.com/articles/intuitive-understanding-of-eulers-formula/

Jajal, T. (2018, May 30). Distinguishing between Narrow AI, General AI and Super AI. Retrieved on January 18, 2019, from https://medium.com/@tjajal/distinguishing-between-narrow-ai-general-ai-and-super-ai-a4bc44172e22

Kaku, M. (2011, 15 march). Physics of the Future: How Science Will Shape Human Destiny and Our Daily Lives by the Year 2100. Retrieved on 4 november 2018, from https://www.goodreads.com/work/quotes/13358451-physics-of-the-future-how-science-will-shape-human-destiny-and-our-dail

Keras. (n.d.). Optimizers - Keras Documentation. Retrieved on 5 december 2018, from https://Keras.io/optimizers/

Kingma, D., & Lei Ba, J. (2015). Adam: a method for stochastic optimization. Retrieved on 5 december 2018, from https://arxiv.org/pdf/1412.6980.pdf

Maini, V. (2018, 16 august). Machine Learning for Humans, Part 3: Unsupervised Learning. Retrieved on 8 november 2018, from https://medium.com/machine-learning-for-humans/unsupervised-learning-f45587588294

McLachlan, H. (2016, August 11). Is every human voice and fingerprint really unique? Retrieved on January 20, 2019, from https://theconversation.com/is-every-human-voice-and-fingerprint-really-unique-63739?sr=1

Merriam-Webster. (n.d.). Definition of Artificial Intelligence. Retrieved on 7 december 2018, from https://www.merriam-webster.com/dictionary/artificial%20intelligence

Moore's law | computer science. (n.d.). Retrieved on January 18, 2019, from https://www.britannica.com/technology/Moores-law

Multi-Class Neural Networks: Softmax | Machine Learning Crash Course | Google Developers. (2018, 1 october). Retrieved on 5 december 2018, from https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax

Reddi, S., Kale, S., & Kumar, S. (2018). On the convergence of Adam and beyond. Retrieved on 5 december 2018, from https://openreview.net/pdf?id=ryQu7f-RZ

Reedy, C. (2017, October 16). Kurzweil Claims That the Singularity Will Happen by 2045. Retrieved on January 18, 2019, from https://futurism.com/kurzweil-claims-that-the-singularity-will-happen-by-2045/

Sharma, A. (2018, 21 june). Understanding Activation Functions in Neural Networks. Retrieved on 5 december 2018, from https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

Shivkumar, A. (2015, 29 october). Why would adding more neurons per hidden layer in a neural network hinder convergence within the actual training set? [Forumpost]. Retrieved on 7 december 2018, from https://www.quora.com/Why-would-adding-more-neurons-per-hidden-layer-in-a-neural-network-hinder-convergence-within-the-actual-training-set

Stanford University. (n.d.). Unsupervised Feature Learning and Deep Learning Tutorial. Retrieved on 16 november 2018, from http://deeplearning.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

Tensorflow. (2018, 20 november). tf.train.AdamOptimizer | Tensorflow. Retrieved on 5 december 2018, from https://www.Tensorflow.org/api_docs/python/tf/train/AdamOptimizer

Torres, J. (2018, 26 september). How Do Artificial Neural Networks Learn? Retrieved on 12 november 2018, from https://towardsdatascience.com/how-do-artificial-neural-networks-learn-773e46399fc7?gi=8b0f53201f9

Ullah, Z. (2012). Early Computer VS Modern Computer: A Comparitive Study and an Approach to Advance Computer. Retrieved on 4 november 2018, from http://www.academia.edu/29957529/Early_Computer_VS_Modern_Computer_A_Comparitive_Study_and_an_Approach_to_Advance_Computer

University of Iowa. (n.d.). Nature versus nurture of voice | voice-academy. Retrieved on 16 november 2018, from https://uiowa.edu/voice-academy/nature-versus-nurture-voice

Voskoglou, C. (2017, May 5). What is the best programming language for Machine Learning? Retrieved on January 17, 2019, from https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7?gi=80bad449c44d

Walia, A. (2018, 24 august). Activation functions and it's types-Which is better? Retrieved on 5 december 2018, from https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f?gi=2dde68909f4dhttps://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

Wikipedia contributors. (2018, 11 november). History of neurology and neurosurgery - Wikipedia. Retrieved on 11 november 2018, from https://en.wikipedia.org/wiki/History_of_neurology_and_neurosurgery

Wikipedia contributors. (2018, 15 november). Deep learning - Wikipedia. Retrieved on 16 november 2018, from https://en.wikipedia.org/wiki/Deep_learning

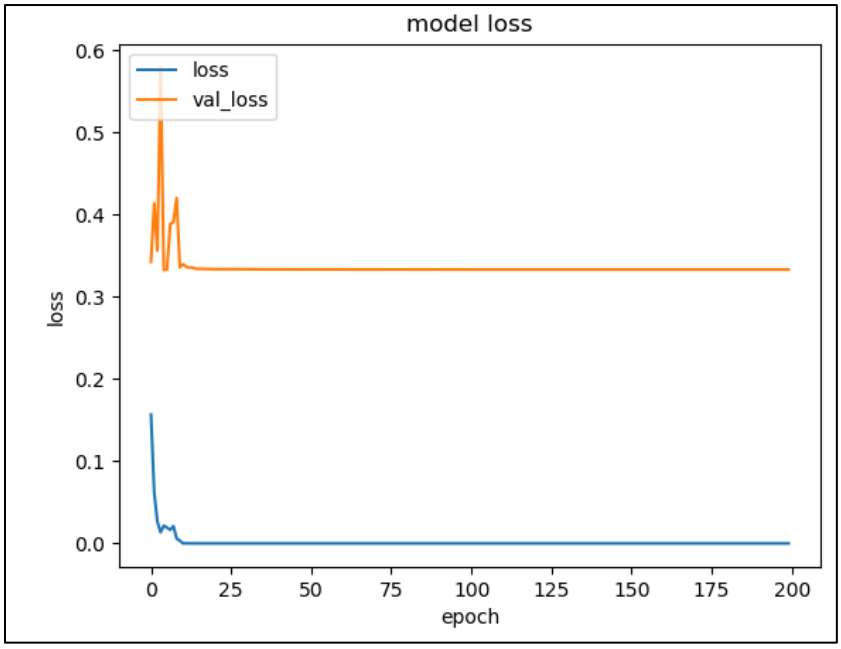
Wikipedia contributors. (2018, 16 november). Fourier transform - Wikipedia. Retrieved on 17 november 2018, from https://en.wikipedia.org/wiki/Fourier_transform

Wikipedia contributors. (2018, 16 november). Human voice - Wikipedia. Retrieved on 16 november 2018, from https://en.wikipedia.org/wiki/Human_voice

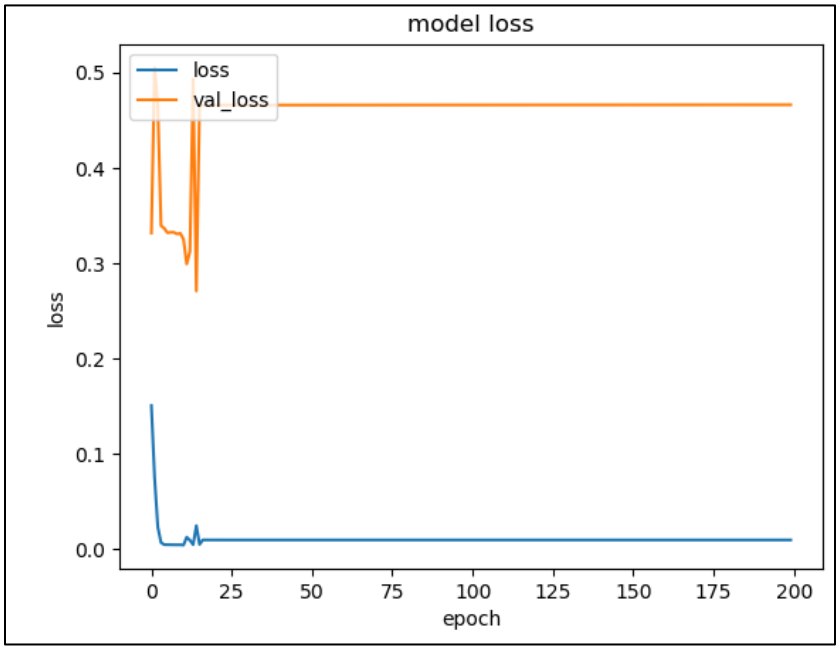
Wikipedia contributors. (2018, 16 november). Microphone - Wikipedia. Retrieved on 17 november 2018, from https://en.wikipedia.org/wiki/Microphone

14. Attachments

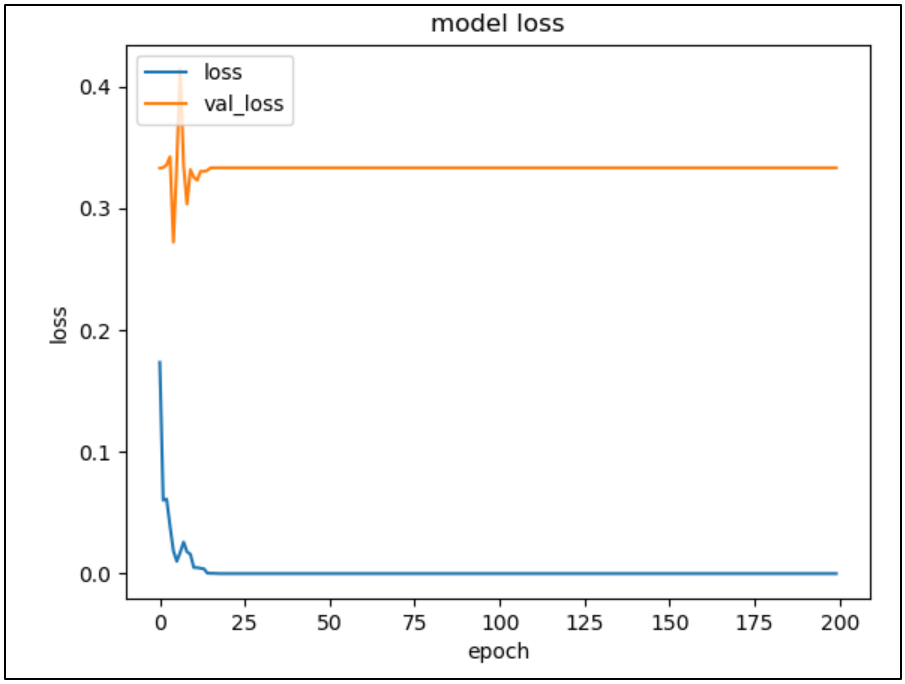
14.1 Section A: Network 1



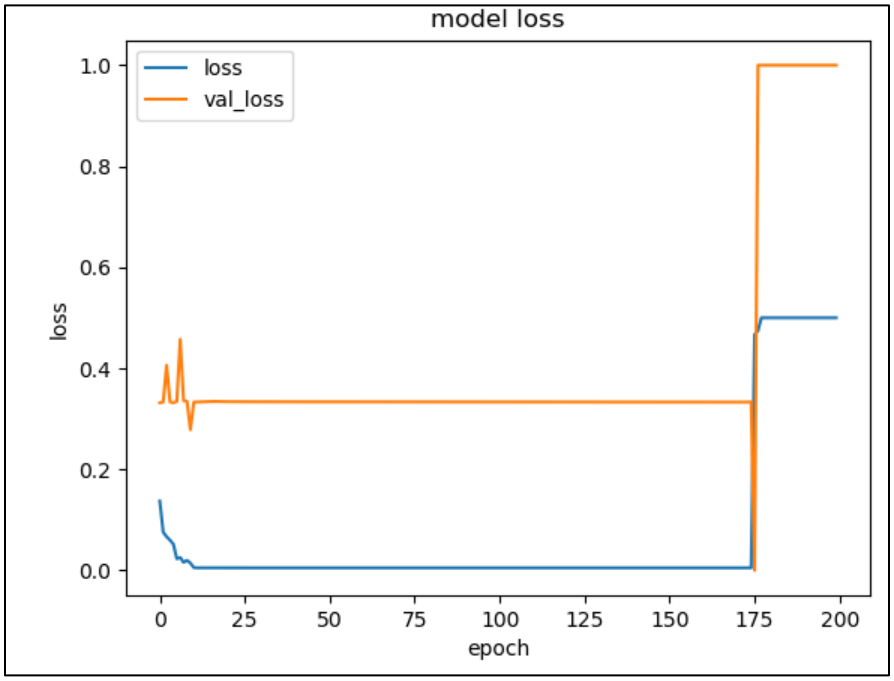
Attachment A 1: Loss and val_loss plotted over 200 epochs



Attachment A 2: Loss and val_loss plotted over 200 epochs

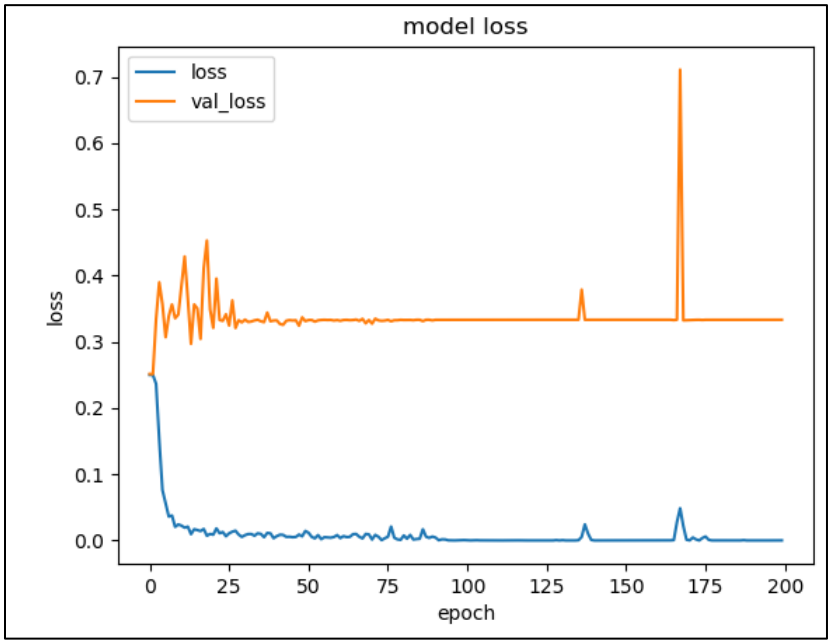


Attachment A 3: Loss and val_loss plotted over 200 epochs

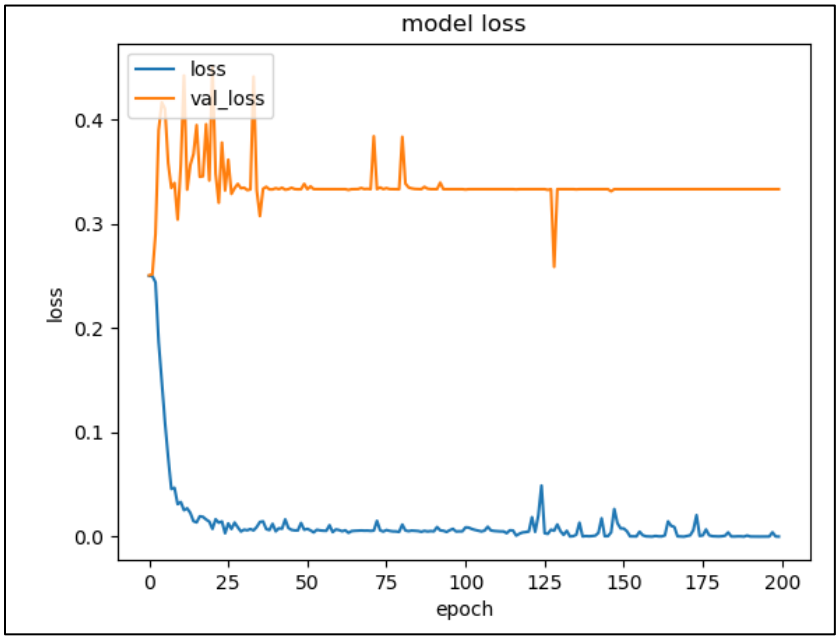


Attachment A 4: Loss and val_loss plotted over 200 epochs (predictions were wrong with this configuration as well)

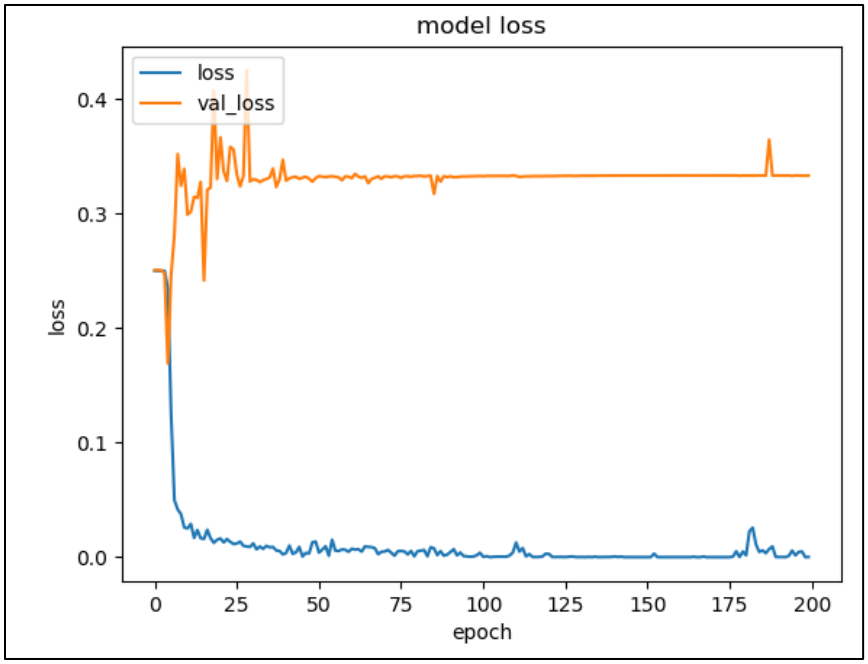
14.2 Section B: Network 2 with two speakers



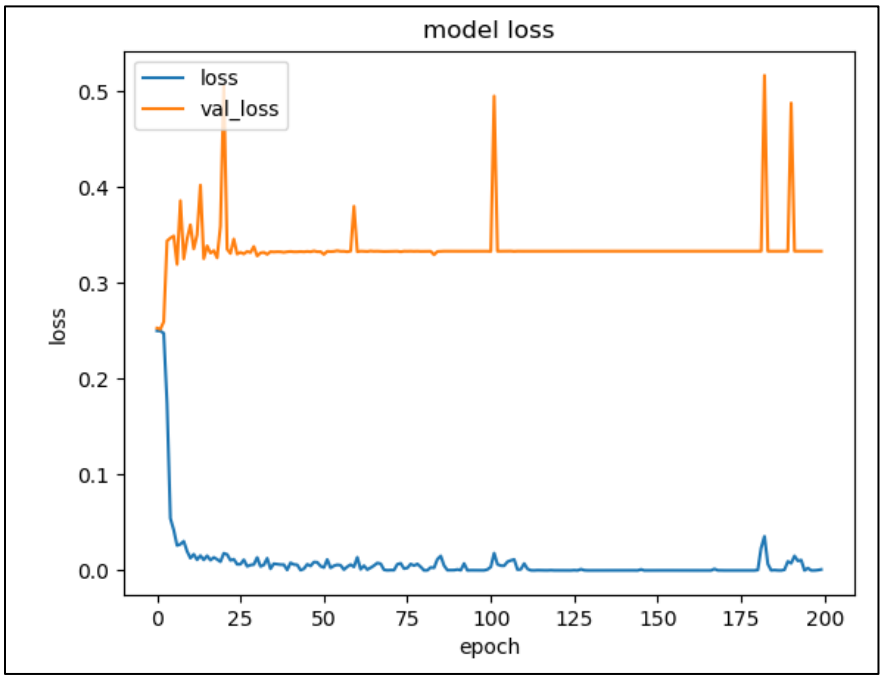
Attachment B 1: Loss and val_loss plotted over 200 epochs



Attachment B 2: Loss and val_loss plotted over 200 epochs

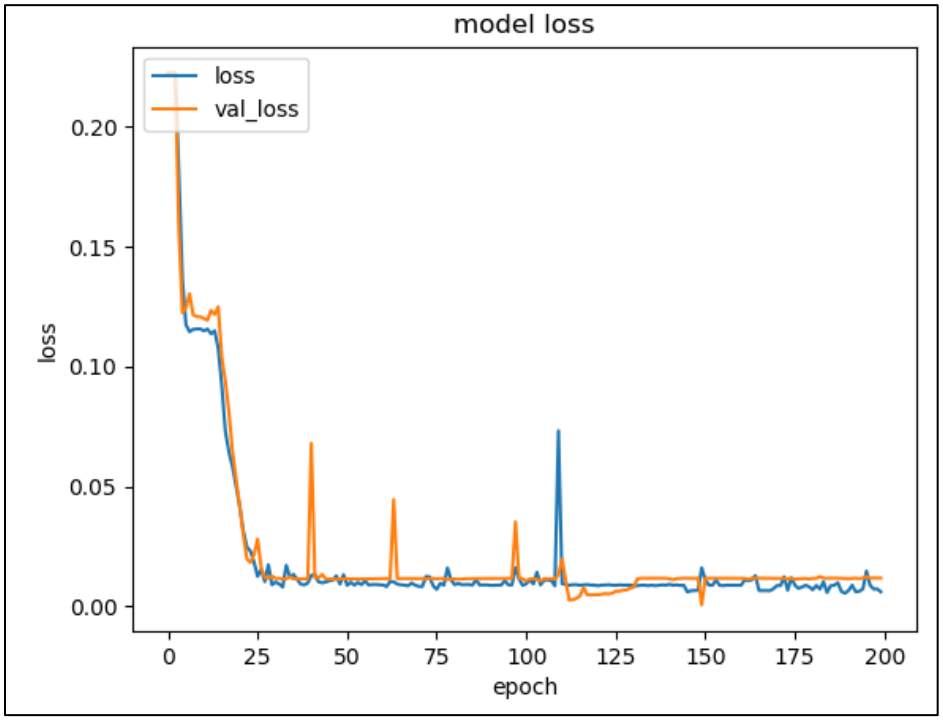


Attachment B 3: Loss and val_loss plotted over 200 epochs

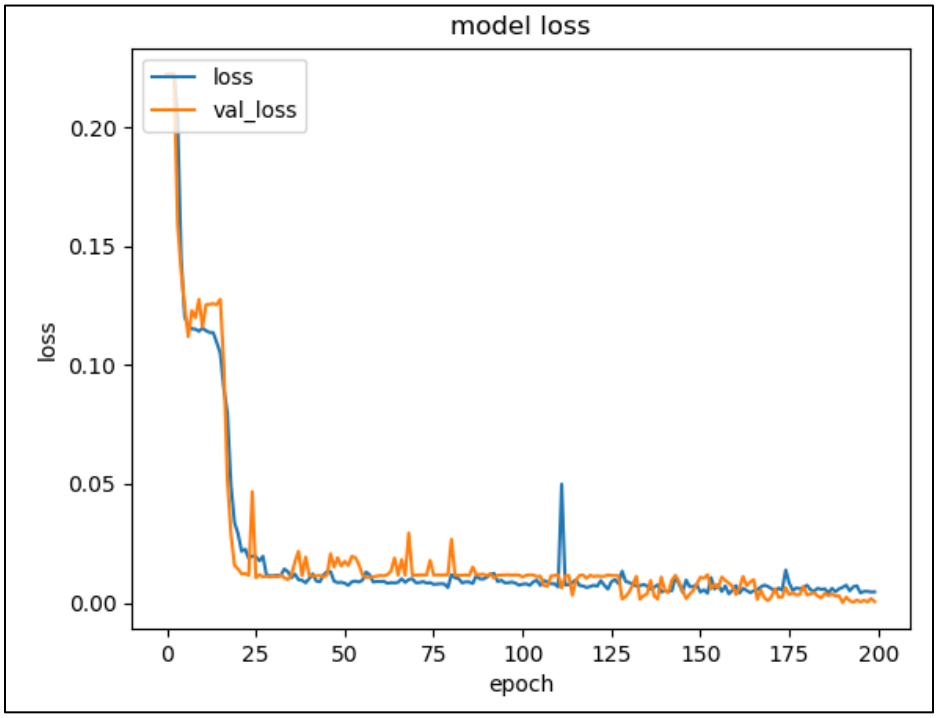


Attachment B 4: Loss and val_loss plotted over 200 epochs

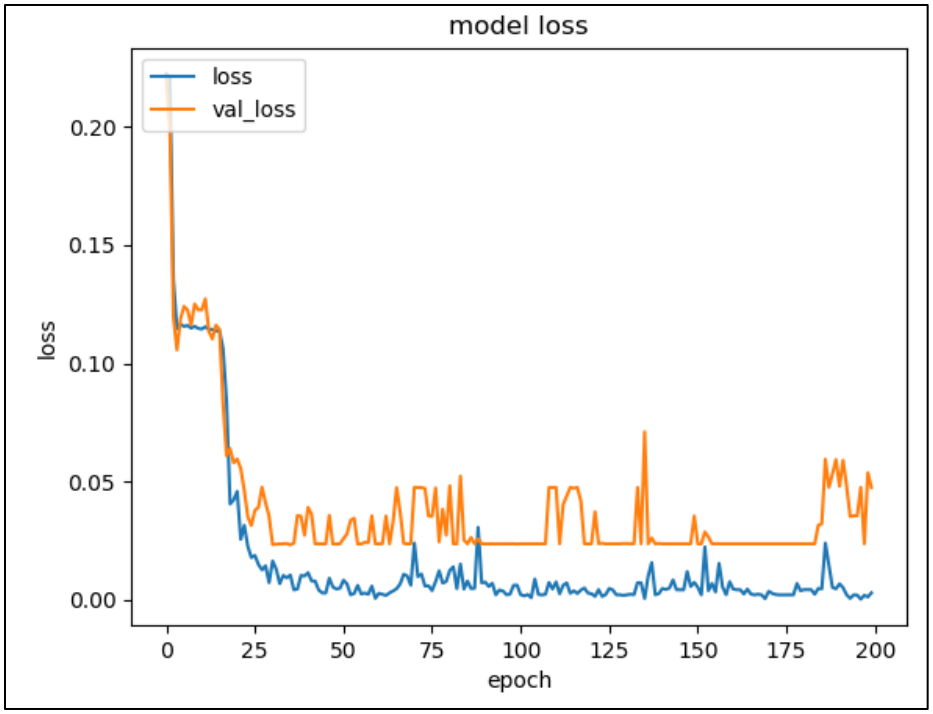
14.3 Section C: Network 2 with three speakers (50 neurons)



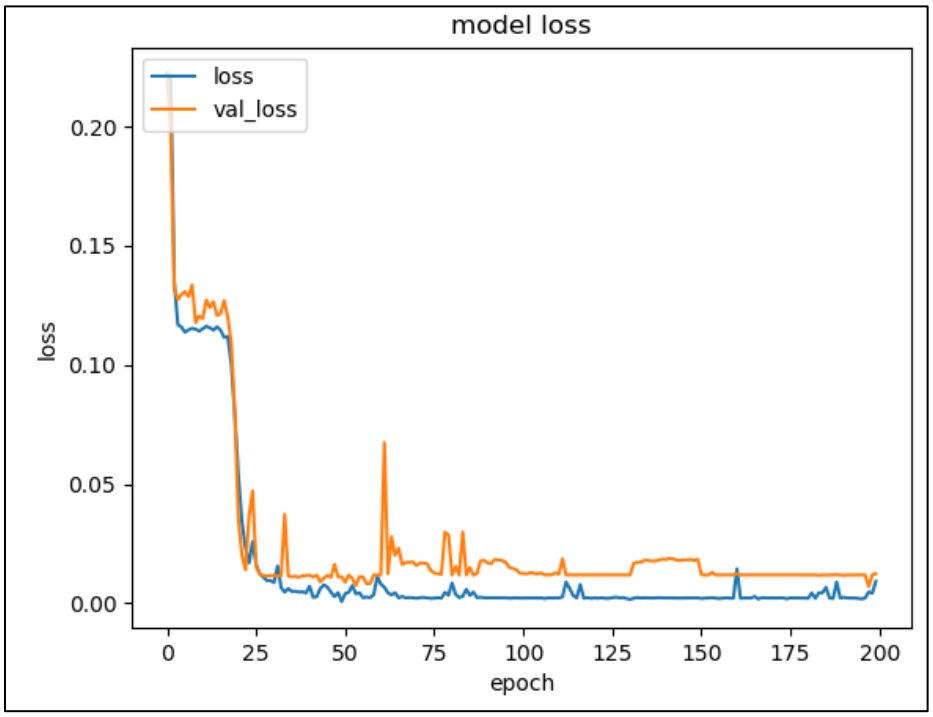
Attachment C 1: Loss and val_loss plotted over 200 epochs with 50 neurons



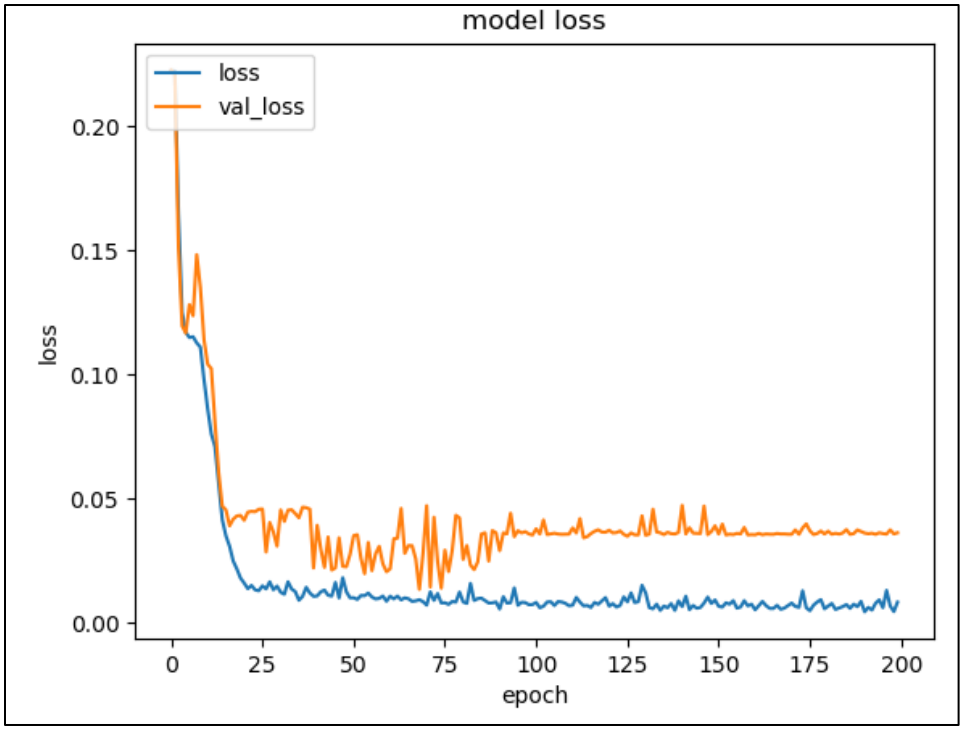
Attachment C 2: Loss and val_loss plotted over 200 epochs with 50 neurons



Attachment C 3: Loss and val_loss plotted over 200 epochs with 50 neurons

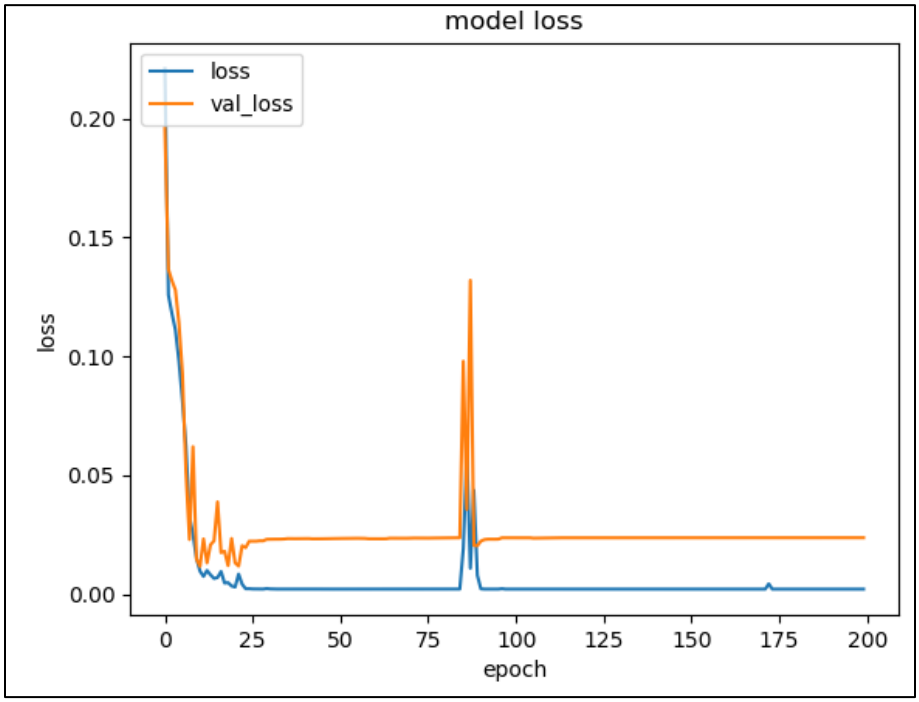


Attachment C 4: Loss and val_loss plotted over 200 epochs with 50 neurons

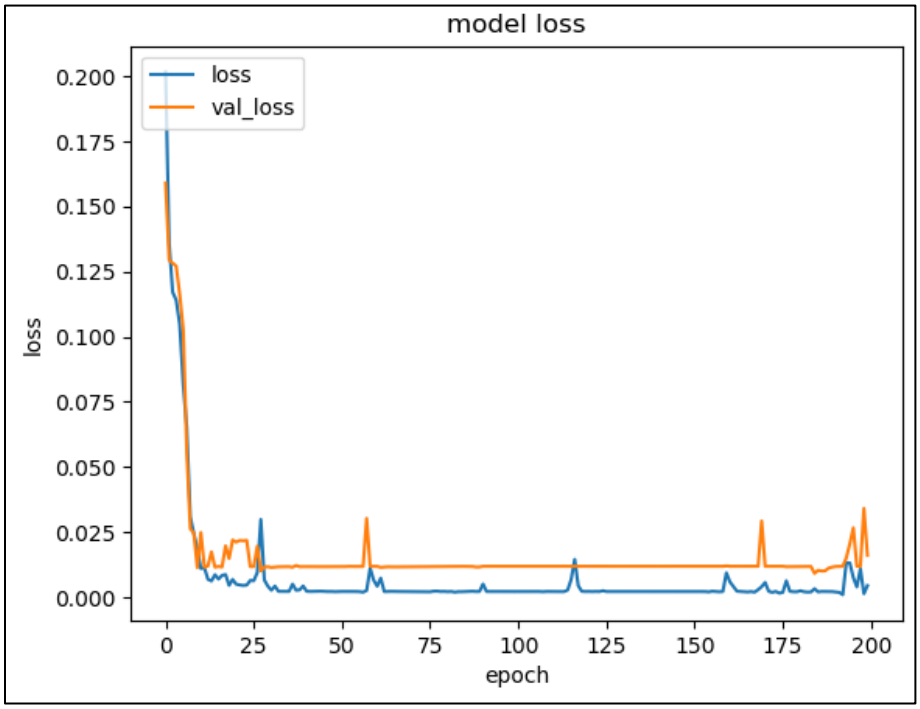


Attachment C 5: Loss and val_loss plotted over 200 epochs with 50 neurons

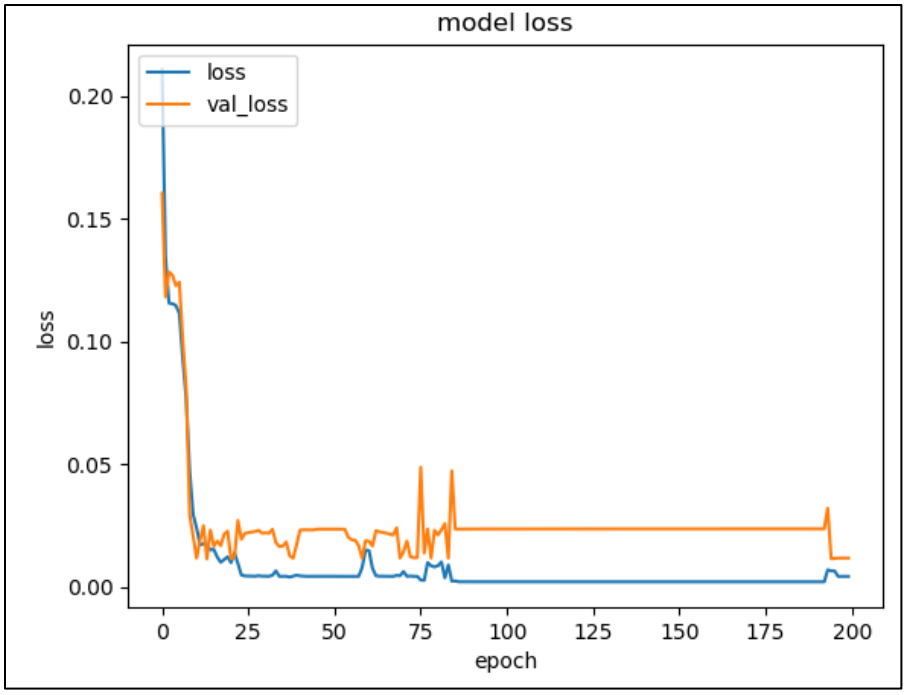
14.4 Section D: Network 2 with three speakers (100 neurons)



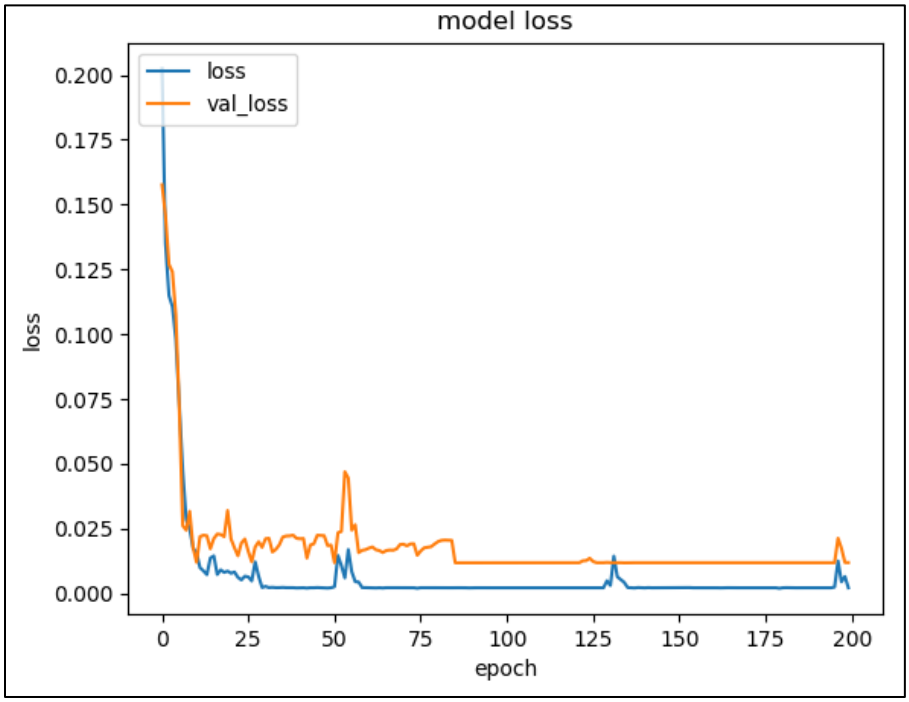
Attachment D 1: Loss and val_loss plotted over 200 epochs with 100 neurons



Attachment D 2: Loss and val_loss plotted over 200 epochs with 100 neurons

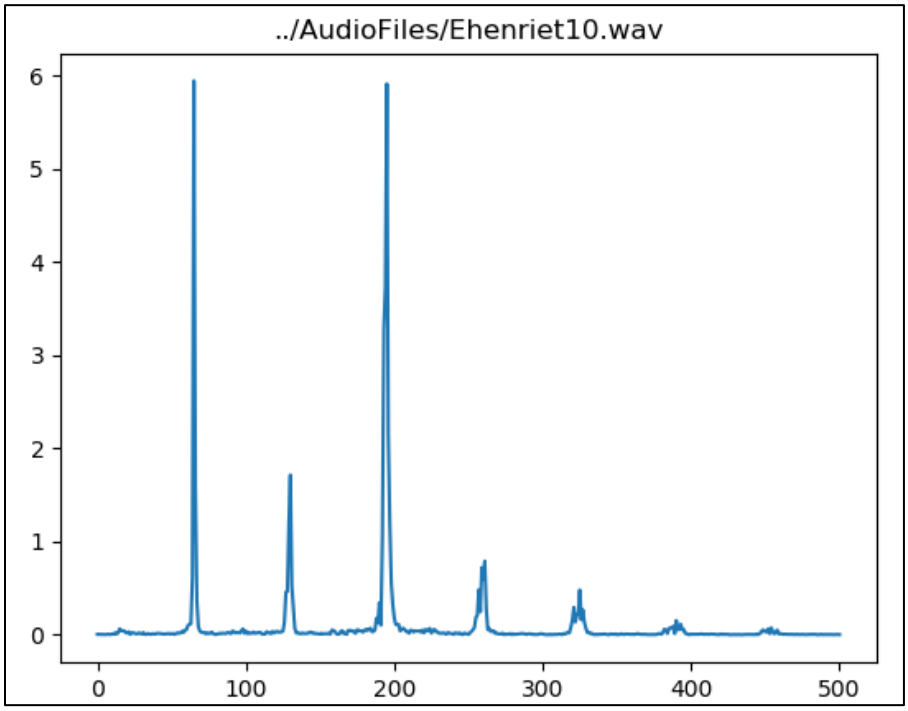


Attachment D 3: Loss and val_loss plotted over 200 epochs with 100 neurons

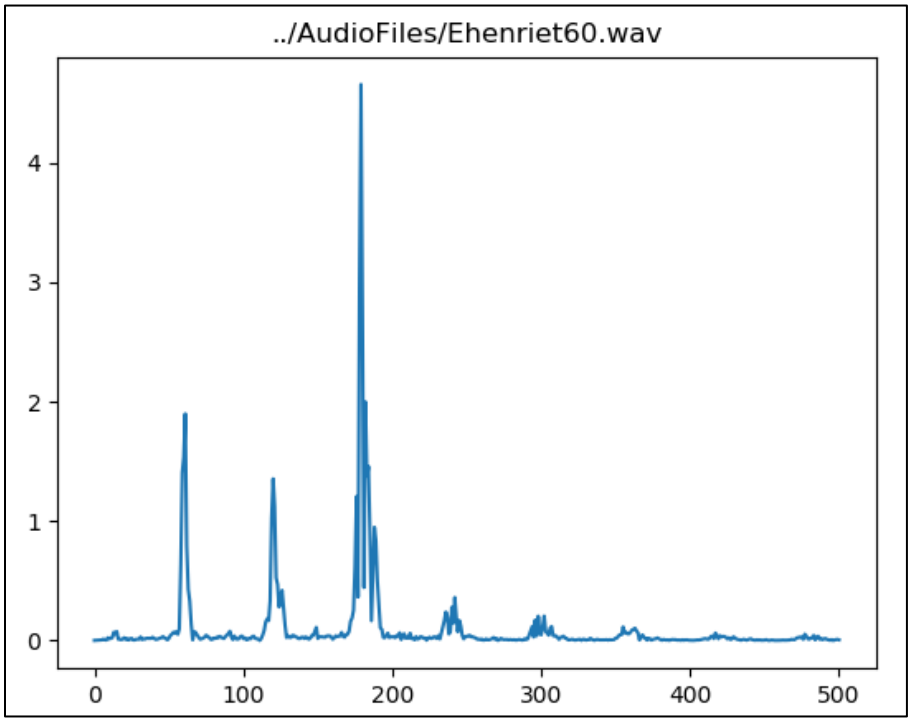


Attachment D 4: Loss and val_loss plotted over 200 epochs with 100 neurons

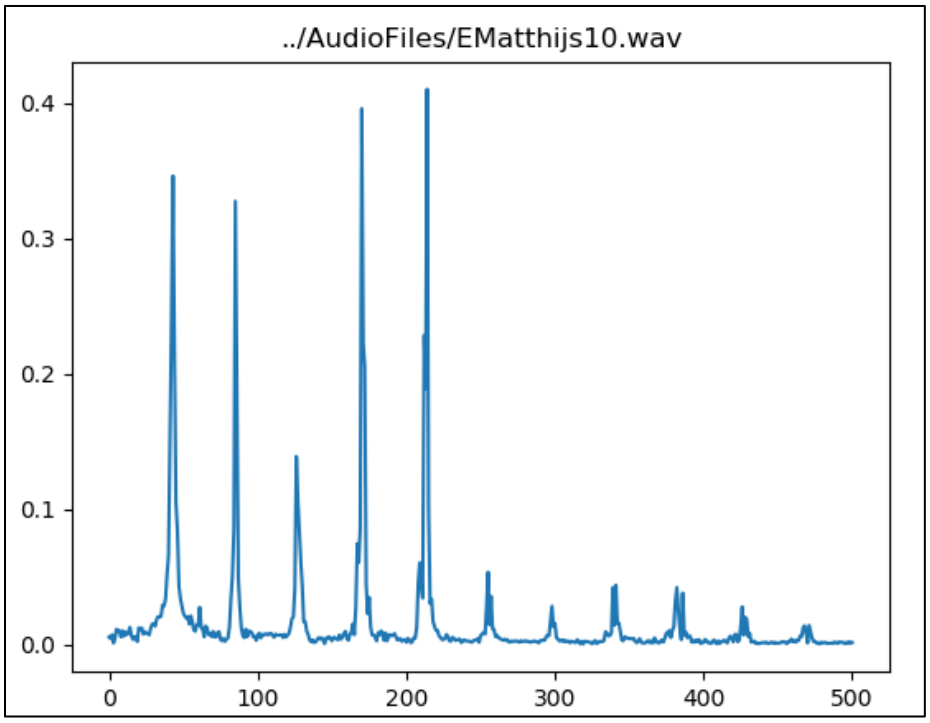
14.5 Section E: Fourier graphs



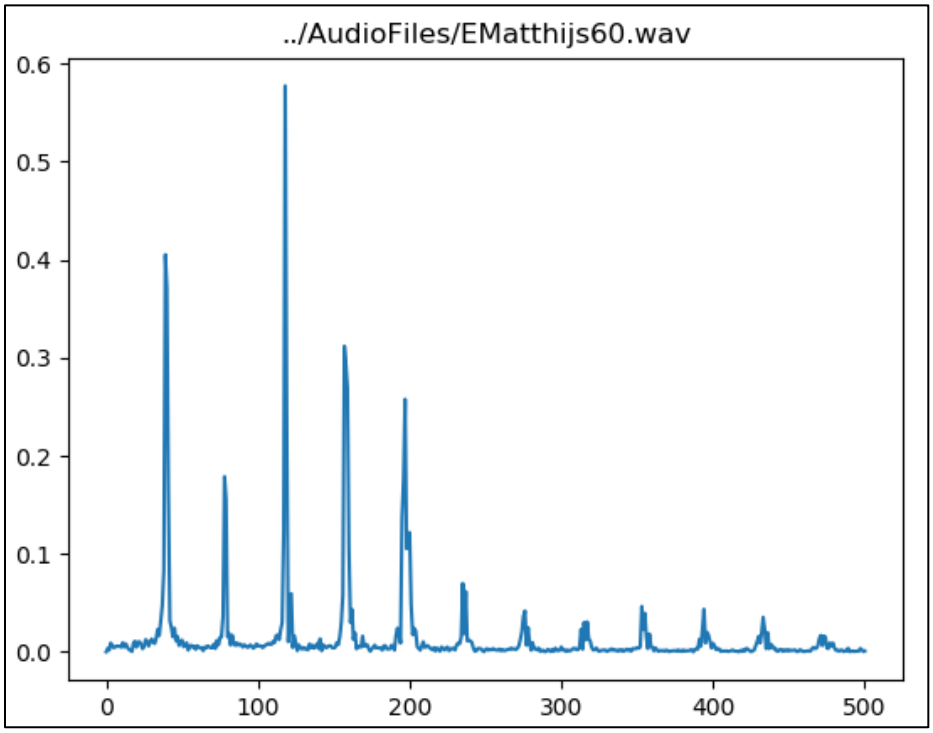
Attachment E 1:Fourier Henriet 10



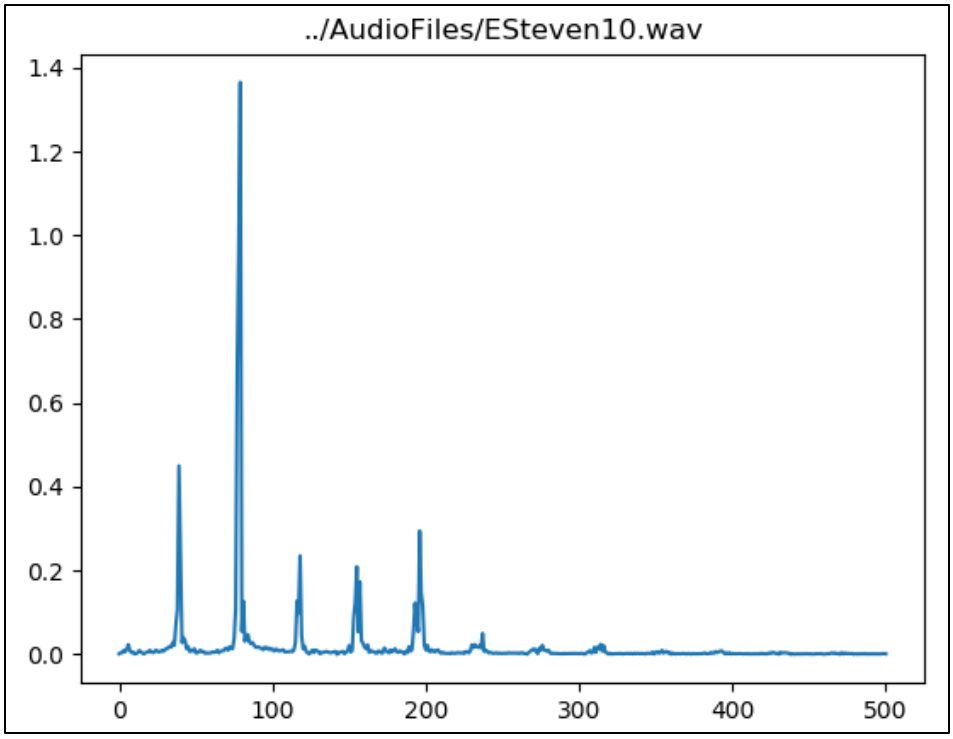
Attachment E 2: Fourier Henriet 60



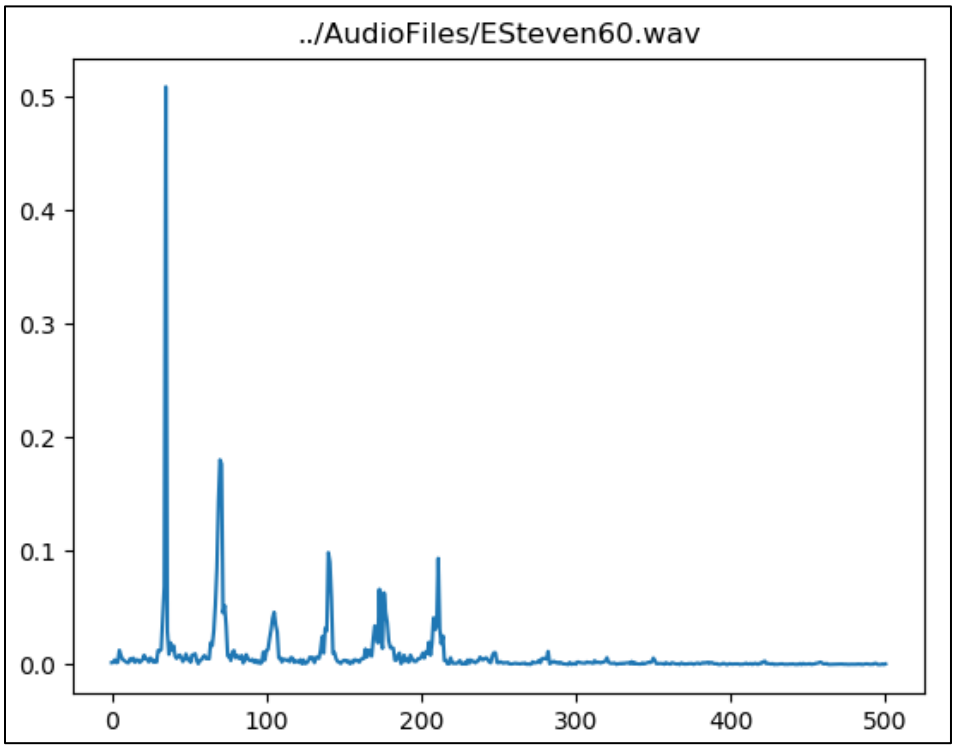
Attachment E 3: Fourier Matthijs 10



Attachment E 4: Fourier Matthijs 60



Attachment E 5: Fourier Steven 10



Attachment E 6: Fourier Steven 60

14.6 Section F: Philosophy and ethics

Currently, artificial intelligence is mostly portrayed in a bad way in the media. Famous people like Elon Musk and Stephen Hawking have quotes that warn about the dangers of AI. Although big improvements are made every day, the ethics of AI should not have the priority on what we should solve today, even though it should be discussed eventually. Problems we face like climate change should be first on the list of worrying about. AI can seem like something scary or threatening if someone has little knowledge of it. Instead of fearing the change, taking a look at the positive side will show that AI can help a great deal with the evolution as humans.

At the moment, great minds are working together in order to achieve great heights with artificial intelligence such as the recognition of diseases, assistance of elderly and the prediction of proteins. If used for a good cause, we believe AI can help humanity go a far way.

The capabilities of AI are increasing every year. artificial intelligence is beating humanity in an increasing number of fields, especially in computational and analytical ones. The world champions in games such as GO and chess have been beaten by AI, and most stock trading is done by computer software. However, this does not mean that the AI is 'smarter' than humans. A computer can beat a human in chess, but it does not have the faintest clue on how to play checkers. These types of AI excel in one thing, and one thing only. They are called 'Artificial Narrow Intelligence', abbreviated as ANI[55]. Currently all the 'intelligent' systems we have created are ANI, programs that excel at one thing, but nothing else.

The next step in AI classification has not been reached by humanity yet. This is 'Artificial General Intelligence', or AGI. An AGI is a program that is as smart as a human being in every field humanity can think of. Theoretical physics, creativity, philosophy, psychology, voice recognition and even computer programming. Everything a human is capable of, an AGI is capable of as well. After AGI has been reached it is but a short step to the next, and final, type of AI, called Artificial Super Intelligence (ASI).

An AGI can do anything a human do, but there is a key difference. Humans are limited by evolution, improvement to the human race can take thousands of years. Theoretically, AI is not limited by this, it could re-program itself at incredible speeds with the same skill as the best programmers in the world.

Right after AGI has been achieved the system would be able to re-program itself faster and better than any human ever could, becoming an ASI. This ASI would be even smarter than the AGI and be able to re-program itself even better. This cycle continues until an exponential amount of growth would occur, where the ASI outclasses humanity by such astronomical amounts we would not have any control over it. This point is called the technological

[55] Jajal, T. (2018, May 30). Distinguishing between Narrow AI, General AI and Super AI. Retrieved on January 18, 2019, from https://medium.com/@tjajal/distinguishing-between-narrow-ai-general-ai-and-super-ai-a4bc44172e22

singularity[56]. What would happen after such a singularity occurs is not clear, as the human mind could not even comprehend what would go on in the 'mind' of such a super intelligent being. It would be like an ant trying to understand calculus without even having the slightest grasp of addition and subtraction.

No reason for despair yet though. Although we are progressing rapidly, this tiny fracture of experience that we have gotten has resulted in us thinking that we are still relatively far from such a singularity. We hope that this paper has not only helped understanding the basics of Neural Networks but also to understand why we think there is no reason for despair yet.

[56] Reedy, C. (2017, October 16). Kurzweil Claims That the Singularity Will Happen by 2045. Retrieved on January 18, 2019, from https://futurism.com/kurzweil-claims-that-the-singularity-will-happen-by-2045/

Ondergetekende verklaart:
Naam: Steven van den Wildenberg
. dat dit PWS eigen werk is;
. dat alles wat overgenomen is uit enige bron voorzien is van een correcte bronvermelding.
Heemstede, 21-01-2019
Handtekening:

Ondergetekende verklaart:
Naam: Matthijs Ates
. dat dit PWS eigen werk is;
. dat alles wat overgenomen is uit enige bron voorzien is van een correcte bronvermelding.
Heemstede, 21-01-2019
Handtekening: