Data Processing for the Cocktail Party Problem

A review of **

Héctor M. de la Rosa Prado

A thesis presented for the degree of
Bachelor in IT for Science

Department Name **
National Autonomous University of Mexico
Mexico
January 21, 2020

Contents

Acknowledgements

Data Processing for the Cocktail Party Problem

Thesis Subtitle

Héctor M. de la Rosa Prado

Abstract

Preface

This work is the dissertation ”Data Processing for the Cocktail Party Problem” in which we explore the current preprocessing methods used in solving the Cocktail Party Problem oriented in audio. It was written to fulfill the graduation requirements for the bachelor degree in IT for Science.

Chapter 0

Introduction

0.1 The Cocktail Party Problem

Imagine being in a public area with one of your friends. As you talk about your work and what you learned in school there are people around you making noise. They're also talking about their lives, what they plan on doing in the mall or the movie that just came out. Kids laughing with their parents talking, nearby traffic with car horns flaring in the distance. If you wanted to, you could hear them, but instead you ignore it.

You continue to listen to your friend, regardless of the noise in the background. You have no problem paying attention until suddenly you hear your name being mentioned in the crowd. For some reason, somewhere, someone said your name. You look around to see a familiar face walking towards you...

What we described in the short and imaginary story above is what is called the "Cocktail Party Effect". This effect was first coined by Cherry Colin[1] where he proposed a series of tests that would measure the limits of a human's ability to listen to a specific voice under different circumstances. The problem of getting a machine to do this same task was called the Cocktail Party Problem by Cherry.

The cocktail party problem is one of the biggest unsolved problems in computation. The short and imaginary story above gives us an example of how we, as humans, solve this problem in a seemingly effortless way. Thanks to evolution, humans can it so effortless, in fact, that most people don't even stop to appreciate how complicated the task actually is.

Its difficulty, however, is not the reason it is so widely studied. In fact, this small problem seems to be the barrier that has kept us from advancing in the automation of speech related tasks, or at least in the way we would like. The cocktail party is not only present in sound, but in just about any signal processing problem, from medical scanning to telecommunications[2], noise always seems to find its way into our sensors.

That is why a general solution to this problem will not only allow us to improve greatly in audio related tasks, such as speech recognition, transcriptions, audio classification, and audio/speech enhancements, but also in various fields like medical analysis and seismology. The reach of these advancements also promises a wide range of new technologies shortly after.

This is why, in this work, we will attempt to take a dive into how we can leverage the current improvements in artificial intelligence in tackling the problem. Ever since the latest boom of deep learning [3]

Even though, as we saw before, the problem generalizes beyond speech, we will limit our research to stick with voiced data. This is due to the complexity in examining signals as a whole, and because of the focus that most methods currently have on the subject.

Given how the difficulty of the problem lays much beyond my current limits as a researcher we will not show. We will accept limiting our problem greatly for practical reasons, in an attempt to get insight on . This also includes a review of how traditional methods attempt to solve the problem.

0.2 History

Speech separation has been a problem that researchers have been interested in for years. So much so that the problem was formulated decades ago by Colin Cherry[1]. In his famous paper he gives an example of the task with a conversation in a Cocktail party, giving the problem its name. Over 65 years later one could argue that progress is just now being made in the area, mostly due to the advancements in general learning algorithms, like deep learning, giving an edge in unstructured data analysis.

0.2.1 Segmentation vs Attention Problem

Many studies** have been used to divide the problem into more manageable tasks, many of which were included in the experiments made by Cherry himself. Thanks to this, the problem has evolved quite a bit since it was first formulated, deviating from its original definition and including all types of noise, not just speech. As McDermott mentions in his article[2], the problem has been split in two parts, which together engulfs the problem as a whole. Usually both parts are frequently referred to as the Cocktail Party problem interchangeably.

The first part addresses who or what to pay attention to, and when to change targets. For clarity in this paper, we will refer to this specific part as the Attention problem. To solve this problem we need background knowledge about the person paying attention as well as the context to know what the listener wants to listen to. What's more is that the listener might want to listen to two speakers at the same time, given that they are both saying something important. The difficulty of the problem lies in how subjective it is. Because of this, it is practically impossible to model and automate the process.

The second part refers to actually separating the sounds, or also known as "sound segregation". This part will be referred to as the "segregation problem. This differs from the previous task because here we look to separate the audio by different sources. We have no need to know which is more important than the other, as long as we can listen to each source independently. This is the process that has received more attention in the last couple of years, since it is easier to model. That said, it is still an ill-posed problem (more on this later) which means that traditional methods have a hard time.

In this thesis we will only be addressing the segregation problem, trying to separate an audio by the different sources of noise it contains. This problem has quite a bit of research ** but still has a long way before we can actually consider it to be solved. One of the reasons was just mentioned, but it's also due to the fact that the problem, under certain conditions, becomes impossible to model. This leaves us with a few options, we either limit the problem to certain use cases, or we facilitate the problem adding more information. We will consider both of these actions to be "constraints", as one limits the conditions that could be solved while the other limits possible solutions, requiring more information.

0.2.2 Inverse Problems

An inverse problem refers to reconstructing the source from a generated set of components. In this case, we want to reconstruct the separated audios from a generated mixed audio. Inverse problems are usually classified by how "Well-Posed" it is, which is also generally used as an indicator of how difficult the problem could be. Many examples of Inverse problems that extend to signal processing include medical imaging, oceanography, astronomy and geophysics. [InverseProblems]

An inverse problem has 4 components, this includes a measurement operator (MO), injectivity or regularization of the MO, knowledge on noise and the model and finally a numerical implementation. Some of these components are then separated into different sub-catagories, but we will only mention the ones relevant to this work. We address the components based on Guillaume Bal's definition [InverseProblems].

Measurement Operator

- 1
- 2
- 3

Injectivity

- 1
- 2
- 3

Noise and Model

- Noise is practically inevitable in audio, extending from white to brown noise and most times even “structured” noise. This could come from background music or oncoming traffic. When we record in a crowded room the noise changes too, leaving a sound that we can relate with a busy street.
- The measurement will also have errors, but these are less relevant to us. In audio compression, there is always a slight error from the original source to what we hear later on. This, however, is such a small change that we as humans cannot perceive it, at least not without the help of special tools.
- **

Numerical Implementation

For this project we will be using an optimization method, comparing it to previous linear systems. Although these methods were usually considered fairly inexpensive, we will be using neural networks, which are both expensive computational as well as in the data sense.

0.2.3 Ill-Posed Problems

Let’s imagine that we have an audio with two sources that generate noise. One is a dog, and the other is a human. Both audio have different structures and can be separated by a person with quite little effort. We could even separate the audios using certain statistical methods**. Sadly the Cocktail Party includes problems much more complicated than the one we just mentioned. A wide variety of audios include at least two people talking. This is harder to separate, sometimes even for humans, especially if the two individuals have similar voices and speak in a similar manner.

The difficulty of the Cocktail Party problem is that it is Ill-Posed. The terms Ill and Well-Posed problems was first coined by Jacques Hadamard, a French mathematician born in the mid 1800’s. According to his definition, a problem is considered to be Well-Posed if:

- For any input there is a solution
- That solution is unique
- The solution is “stable” in the input space (continuous)

If one of these three conditions were not met, the problem was considered Ill-Posed. At the time, it was considered necessary for any mathematical problem in physics and tech to be formulated as a well-posed problem. This meant that, for practical uses, studying Ill-Posed problems was useless[Ill-Posed].

Now, however, Ill-Posed problems are very common, in fact it is considered rare that a inverse-problem should be well-posed. It has come to the point where researchers seemingly compare how “Ill-Posed” a problem may be[InverseProblems]. These extends to Well, mildly Ill and severely Ill-Posed problems.

0.2.4 Constraints

If we continue with our thought experiment from before, what would happen if instead we had two machines that emit static noise of the same type, same volume similar position etc. This would be practically impossible to separate without knowing more about the noise being emitted to try and find some structural difference in the two. Not only that, but the problem wouldn't be useful in this broader sense. Why would you want to separate noise from noise? The situations where we want to solve the problem don't require us to tackle a perfectly generalized version of it. This is why it is common to set constraints to the problem before beginning to train a model.

As mentioned before, this work is about speech separation, so it is fair to assume that the audio will have at least two people talking. Another constraint we will add is that we will only focus on english speakers, leaving a more general model for another project. As we will see below we will also be using the Common Voice dataset. This dataset is publicly available to make similar models and consists of almost 40000 voices in more than 1000 hours of recorded english voice in different accents.

We could also consider the number of speakers as a constraint, since having only 2 speakers simplifies the problem greatly. During these tests** we will try to make a general model that can split the different speakers even when mixed with an arbitrary amount of voices. Clearly there will be a limit given how 100 voices will more than likely be worse than gibberish, but this arbitrary model could help us separate audio even when unexpected speakers enter the conversation.

Another constraint that could be considered is that we only need to listen to one voice. If you use a hypothetical "perfect" model to separate voices and listen to your friend in a crowded cocktail party, you have little to no interest in listening to other people's conversations. Although this could definitely be easier and give better results, we go back to what we mentioned before when we introduced the cocktail party. This constraint in reality mentions a problem with attention, which for now we are considering a different problem.

0.3 Document Structure

In this project we will separate our investigation into three parts. The first part will review the current literature on the problem, focusing specifically on current processing methods. We then explore how previous studies used these methods and the possible affects this has on the results. This will be split into 4 Chapters.

Chapter 1 will give a short rundown on the current data processing techniques. We will explain why data preprocessing is important and what problems occur without it, focusing on audio tasks. This chapter is divided into two parts, the first will touch on feature extraction for audio. The second part will touch on normalization.

For feature extraction we will need to learn about spectrograms. The spectrograms can be separated into two commonly used categories: linear and mel-bin. Both spectrograms represent audio clips into a 3-D transformations, similar to images. A linear spectrogram has a linear frequency dimension while mel-bins represents frequency in logarithmic scale.

Chapter 2 will reference one of the problems we currently have in Generating Audio. Here we explain why generating audio directly isn't always possible and what the modern solution for these problems currently looks like. The critiques given to these current solutions will be based on the literature and on my own personal experience.

Chapters 3 and 4 will talk about recent projects. Although these projects do not tackle the problem in the same way, they have reasonably good quality in audio. Because of this, we will use these studies as a reference to what practices could be used to provide these results. Later on we will see that one uses vocoders and the other changes the training target.

At the end of these chapters we will state the pros and cons of each methods and briefly show the train of thought that lead to the experiments that we can find in part 2.

Part two contains the methodology that was taken in this study. In this we hope to explain the experiment well enough to be replicated in future studies. For this we provide a context of the algorithm and how each process was implemented. As we will learn later, the implementation of some of these algorithms which varies in the different libraries affect the outputs similar to the parameters.

Finally the third and final part of this text will give the results to the project. In this we will explain what the quantified errors represent and a quick qualitative assessment of the audios. ***As a reminder, we do not plan on building a model that competes against the current SoTA. ***Instead we will attempt to find input and target creation techniques that improve the quality of the generated audio.

The final part of this thesis contains the appendix and the bibliography. In the appendix we give most of the theoretical background necessary to understand the methods used in our research. This includes subjects ranging from machine learning, signal processing, the Fourier and wavelet transform and a small usage guide for the source code and where to find it.

Part I

Literature Review

Chapter 1

Audio Data Preparation

Transforming and processing data is an important part of deep learning. Without this step, most models widely under perform in their tasks while others will not work at all. Even tasks like categorizing involve some sort of preprocessing of data like one-hot encoding.

This stays true for audio. In fact it could be more true than many other machine learning problems. Most audio files come with different parameters, lengths, file formats, volumes and sampling rates. Finding the correct parameters and feature extraction methods for audio has led to quite a few barriers in the area.

Although there are still many unanswered questions when it comes to preprocessing audio, the literature still offers some hints on which approaches could work better than others. One example of this is data representation. In the digital world, audio can be represented by a sequence of numbers and usually a sample rate. With a fixed sample rate, we can consider the sequence to be a simple time series.

Most deep learning processes involving time series make use of recurrent neural networks that receive one or more of these data points, with audio. However, at it turns out, this is not enough. Even simple binary audio classification has a hard time achieving promising, let alone reasonable results. ***

This is not the only problem we face. Apart from that, it is also extremely slow to process all of these data points. If we were to consider a small 10 second audio clip at a 22050 sampling rate (half of what is normally used) this will still give us a 220500 data point time series. Generative models suffer an even greater overhead, which is a great critique to models like WaveNet which take a point by point generation approach. ***

Because of this, part of the research implies exploring different parameters. Even with the limited study on these values, we can still sometimes rely on expert recommendations to give us an idea of what parameters would reveal better results.

For now, we will review data transformations necessary to input into a neural network. After that we will explore audio data preparation techniques, including data wrangling and feature extraction.

1.1 Data Wrangling

In deep learning, data wrangling, also known as data munging, consists of changing the data in a way that is convenient to be processed by the model. *** Usually includes taking care of missing, noisy, and inconsistent data in out dataset. This process can be manual, but because machine learning and, more specifically, deep learning models tend to have large datasets, it is better to automate the process to avoid wasting time and remove possible human errors though repetition.

That is why various techniques have been developed to solve these problems automatically. An important and sometimes difficult decision can be choosing which data wrangling method to use. Thankfully most of these are easy to implement with any modern scripting language, which allows us to test and see which methods work better than others empirically.

One example of how data wrangling could look like is making sure that the data all contain the same dimensions. This is easier to demonstrate with images, where we tend to resize the images in order to see them in the same shape. Below we can see the difference between an image with *** dimensions and the same image resized to ***.

***

A good question to be asked at this point is, what happens if we don't wrangle our data? For cases in which the data is missing, it's likely we will get an error due to trying to access a part of memory that doesn't exist. Because of this we are obviously force to fix this problem one way or another before being able to use our dataset, even if we just discard incomplete items.

For noisy and inconsistent data, however, we might be able to run our model but receive poor results in training. The noise could be large enough to make it harder for the model to find patterns in the data. For inconsistent data one value could mean different things depending on how the data was collected, leading to confusion trying to optimize our values.

These ideas can be applied to all data, including audio. Audio needs to be resized, normalized and even transformed to similar data types before we can dependably use it in our learning process. We can also find corrupt or incorrect files in the dataset that need to be removed or corrected.

Below we will mention a few of the processes that are used in certain papers and that seem important for most audio related tasks. This includes removing noisy backgrounds, normalizing, resampling, removing corrupt files and clipping/padding.

1.1.1 Sampling Rate

In most audio recordings, the sample rate determines how many data points are in a second of audio. This means that the data points in the time sequence depend on the sample rate being used. If the sampling rate is not correct, or if you change it directly, it will change the speed and quality of the audio. Although at first sight this doesn't seem too bad, if done by hand tends to sound unnatural depending on how drastic the change was.

Because of this, setting up a standard sampling rate for all of the clips in our data set can get a bit complicated. Luckily most datasets set up all of the audios to have the same sampling rate. Sampling rates also tend to have values of 22050, 44100 or 48000, making it easy to get a dataset with the same sampling rate.

Still, for certain datasets, or when constructing you own, you will need to take a different approach. If you want to down sample your audio to a sampling rate that is perfectly divisible by it's original, you can simply skip data points. For example, if you want to down sample 44100 to 22050, you only have to skip every other data point in the time sequence and then set your new sample rate.

Still, it is useful to be able to resample your audio to what ever sample rate you need. To do this, you can use a resampling algorithm, the one most recorded in most public repositories is a poly-phase resampling algorithm. *** The algorithm is not difficult to implement but can also be found in certain libraries that can be used for signal processing, like scipy. ***

1.1.2 Clipping and Padding

Sometimes, the length of the times series isn't what our model was expecting. Even recurrent neural models tend to have a fixed input and just cycle endlessly until the input is finished. Because of this, unless we process the data points one at a time, we could find ourselves with an audio that needs to be padded.

We already mentioned the down sides to using this type of approach, and although it's generative quality is great, we cannot fully depend on these algorithms for all tasks. Another problem is that sometimes we don't have a list of individual audios, but instead only a few extensive audios that need to be processed in smaller clips.

1.1.3 Normalization

In the data time series, the amplitude of the audio wave is set by

1.2 Feature extraction

In machine learning, feature extraction is the process of representing data in a more "understandable" way. In audio this is usually done by applying a short-time fourier transform (stft) as well as other transformations. Next we will give a brief explanation on how the stft works, but first we need to make sure we know what the fourier transform does.

1.2.1 Spectrums

The fourier transform is used to decompose an signal into it's base sine wave components. If, let's say, we have the signal composed of 2 sine waves like below, the fourier transform can used to find that the signals sin(2t) and sin(3t) are present. The second image is of the output, which is called a spectrum, where we can see that the peaks of the plot represent the sine components we are looking for. This way, we can process a signal and replace it with it's components.

There are two problems with using the fourier transform directly into an audio on

1.2.2 Spectrogram

Once we apply the stft to an audio file, we will have what is called a spectrogram. To get an idea of how the two differ, you can see the images below. The first image is the plotted time series of a short audio sample. The second is a plotted image of the spectrogram.

The second image has more dimensions. In fact, the image doesn't exactly give us all of the information because the intensities have to be converted from complex numbers to it's magnitude values. Regardless of this, we still have 3 dimensions only this time they are time, frequencies and intensity.

This new dimension for frequencies is what make analyzing a spectrogram much easier. Here we can clearly see the difference between frequencies and sometimes even make out certain noises, if the image resolution is correct. Below we can see an example of how the "sh" sound would look like.

As you can imagine, if we can distinguish certain sounds from this spectrogram without any expert knowledge, it can be much easier for our model to make predictions. The spectrogram still depends on the parameters we use in the stft. Because there are not many studies that explore these parameters most papers rely on the default parameters for these transforms.

1.2.3 Mel-Spectrogram

Like we mentioned before, using crude audio files for deep learning has not had success in the field. One solution to this is to use feature extraction methods, which "digests" the data into a more learnable representation. Amongst these methods, spectrograms have become the preferred audio representation for Machine Learning.

Part of this is due to the similarities that spectrograms seem to have to images, which has been one of the most extensively researched areas in deep learning. Certain studies even go as far as to transform spectrograms to rgb channeled images to directly use with well known images classification techniques. ***

There are many types of spectrograms, each one including even more parameters which need to be explored and tested. Most of the current research that revolves around correct parameter settings focus on classification, with others focusing on transcription.

This leaves us with a limited idea as to how we should set our parameters for certain studies like the one in this paper.

Chapter 2

The Audio Generation Problem

Chapter 3

Looking to Listen

Chapter 4

Music-Speech Separation

Part II

Methodology

Chapter 5

Libraries

5.1 Librosa

5.2 PyTorch

5.3 TorchAudio

Chapter 6

Implementation

6.1 Dataset

6.2 Model Structure

6.3 Preprocessing

6.4 Targets (Tested)

Part III

Results

Chapter 7

Results

7.1 Initial Dataset

7.1.1 Quantified Losses

7.1.2 Quality Tests

7.1.3 Samples (images and audios)

7.2 Transfer learning

7.2.1 Quantified Losses

7.2.2 Quality Tests

7.2.3 Samples (images and audios)

7.3 Transfer Learning with fine tuning

7.3.1 Quantified Losses

7.3.2 Quality Tests

7.3.3 Samples (images and audios)

Chapter 8

Discussion

8.1 Future Work

Appendices

Appendix A

Time Series

A.1 Signal Processing

A.2 Speech Properties

Appendix B

Transforms

B.1 Fourier Transform

B.1.1 Spectrums

B.1.2 Discrete Fourier Transform

B.1.3 Short-Time Fourier Transform

B.2 Wavelet

Appendix C

Spectrograms

As we saw before, audio can be better represented as mix of different frequencies that define the sound we are hearing. The problem lies in the fact that, in speech, these frequencies change drastically during each conversation, sentence, word and syllable! This means that the perfect spectrogram would need to be a continuous representation of each moment of time and for every frequency.

This is impossible for various reasons, one being the limitations of computational representations, which forces us to make a discrete representation instead. But even if that wasn't the case, the Fourier transform doesn't work on single points in time, instead it works on entire signals. That said, we can use the short time Fourier transform to give us a spectrogram that is discrete in time. Combined with the discrete Fourier transform, we will also have a discrete representation in the frequency axis.

Spectrograms have 3 dimensions, similar to how images are represented, but not exactly. In any given image there are two dimensions that represent space and one that represents intensity. Meanwhile spectrograms have one dimension for time, one for frequency and the last represents the strength and sometimes the phase of the frequency signal at that time. Although some writers define each dimension differently, the standard tends to place frequency as height, time as width and color as signal intensity in a given point.

Although this is a great step forward to giving us a better representation of audio, there are still too many parameters that play a role. Each one of these parameters affects how well the spectrogram can correctly contain the information in the audio. Although there are already well established defaults for most human tasks, it still hasn't been well studied in great part of automated and machine learning tasks.

The exception to this being Speech Recognition, but even so they haven't been heavily studied. There are few papers and datasets that can be used to prove when some parameters are better than others or if there are conservative settings that tend to work consistently in all problems. In the end, some authors question if spectrograms are even the representation that we are looking for for various reasons that we will see in more detail later in Chapter 7.

For now we'll hide our skepticism and focus on the advantages of using spectrograms. The spectrograms we have been referring to so far are what are called linear-spectrograms. Although these are the simplest and easiest to implement, it is not the only one. We will mention the difference between amplitude and decibel spectrograms as well as frequency and mel-bin spectrograms.

C.1 Linear and Log

The values in amplitude are represented in the "color" dimension, giving us the intensities that we see in the spectrogram image. As we saw in the Second Chapter, the amplitude tells us the strength of the signal. In the spectrogram, however, there may also be a phase value. This is because the signals tend to be represented as a Complex number, which can be expressed in cartesian or polar coordinates.

C.1.1 Amplitude vs Decibels

C.1.2 Mel-Bins

C.2 The Phase Problem

C.2.1 Linear

C.2.2 Decibel

C.2.3 Mel-bin

C.3 Phase Retrieval Techniques

C.3.1 Phase Storage

C.3.2 Griffin-lin Algorithm

C.3.3 Vocoders

Appendix D

Machine Learning

"A machine learning algorithm is an algorithm that is able to learn from data."[3] These algorithms search for patterns in the data to discover rules of association that could be used to solve the problem without explicit instructions. This approach turns out to be better, in various task, than previous hard-encoded methods. These older algorithms are usually referred to as traditional methods.

Most of the success of machine learning systems stem from the fact that it is hard to give a perfect definition of something. To achieve this we would be required to tumble into epistemological questions. Most would agree that a definition of a cat would be incomplete without mentioning a tail, yet a cat without a tail is still a cat.

D.1 Supervised vs Unsupervised Learning

Machine Learning algorithms are commonly separated into supervised and unsupervised learning. Even though each of these methods have their pros and cons, most of the current study has focused on developing supervised methods. Part of the reason is because of how data is currently being collected in the real world.

Apart from these two branches of machine learning there is another call "semi" supervised learning**citeSemiSupervised. This approach uses a mix of labeled and unlabeled data items. In real life this allows users to quickly add labels to a dataset. This is done with unsupervised methods, which are later "corrected" by a human user which labels the faulty items.

Even though these tools have proven useful in certain problems, including large scale dataset labeling, they are not relevant to this study.

D.1.1 Supervised Learning

Supervised learning methods, as we mentioned before, use labeled datasets as training data. This allows the algorithm to repeatedly find patterns which could allow it to gain better insights. The training process for these algorithms is similar to a student studying with a mock exam.

In this analogy, our mock exam is the same as our dataset. Even so, just like in real life, we elaborating a mock exam isn't always easy. How do we provide questions similar to the real test, without making it too similar? If you told a class of students that the real exam will be exactly the same as the mock, the students would be less inclined to learn the actual concepts necessary to pass the test (or any similar test). Instead, they might try to memorize the answers knowing that this would be enough.

This same login is what happens when training an algorithm. They don't have a notion of what they are trying to learn but instead what they want to answer. A program is (at least until now) unable to understand that the dataset is just a mock, and that the real test comes after. What's worse, while a normal student would have problems memorizing tests for very long, a computer store these answers with no additional punishment.

D.1.2 Unsupervised Learning

D.2 Advancements

D.2.1 AREAS**

D.3 Problems

D.3.1 Data

D.3.2 Interpretability

D.3.3 Overfitting

D.3.4 Hyper-parameters

D.3.5 Computation

Appendix E

Deep Learning

E.1 Neural Networks

E.1.1 Structure

Inputs

Weights

Bias

E.1.2 Back-Propagation Algorithm

Forward Propagation

Gradients

E.2 Image Processing

E.2.1 Convolutional Neural Networks

E.2.2 Attention

E.3 Segmentation

E.3.1 Medicine

E.3.2 U-Net

Appendix F

Source Code

F.1 Running Instructions

Bibliography

[1] C. E. Colin, "Some Experiments on the Recognition of Speech, with One and with Two Ears," *The Journal of the Acoustical Society of America*, vol. 25, no. 5, pp. 975–979, 1953.

[2] L. Marchegiani, S. Karadogan, T. Andersen, J. Larsen, and L. Hansen, "The Role of Top-Down Attention in the Cocktail Party: Revisiting Cherry's Experiment after Sixty Years," in *Proceedings of the tenth International Conference on Machine Learning and Applications (ICMLA'11)*, (United States), IEEE, 2011.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.