UNIVERSITY PARIS-SACLAY

MASTER'S THESIS

On device Speaker Diarization

Author:
Denys SIKORSKYI

Supervisors:
Dr. Juan CORIA - Ava
Dr. Jean-Michel MOREL - University Paris-Saclay
Dr. Alexey OZEROV - Ava

A thesis submitted in fulfillment of the requirements for the degree of Master's of MVA

September 14, 2023

Abstract

During normal conversations with friends, business negotiations or just working calls, a hearing-impaired or deaf person may encounter difficulties in following what is said. These situations can make people feel neglected or not taken seriously, leading to feelings of inadequacy or loneliness.

The WHO (World Health Organization) predicts that in 2050 there will be more than 1 billion deaf and hearing-impaired people, which is why this matter should not be taken lightly. The recent advances in artificial intelligence have made it possible for new solutions to appear. One of these solutions consists of real-time transcriptions with speaker identification.

However, the neural network models allowing this kind of application are typically too resource-consuming to deploy in edge devices such as mobile phones, which is sometimes required to avoid latency issues due to bad connectivity.

In this work, we focus on reducing the size, and hence inference time, of one of the main components of this solution: the speaker diarization model, whose task is to detect the speakers involved in a conversation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years computing hardware has become more powerful, increasing computation capabilities and allowing it to perform advanced Machine Learning algorithms on edge devices. Such rapid development resulted in the availability of powerful computation units on different platforms like mobile phones or edge devices. Therefore, it became possible to utilize machine learning models on these devices. Following that, the majority of Machine and Deep Learning frameworks incorporated the capability to execute on a wide range of devices, encompassing even mobile phones.

It is worth mentioning, that the previous decade started a rise in demand for executing machine learning models on smartphones with sensitive data like images, audio, etc. Adding into the equation the possibility of running the models on the edge devices results in less resource dependence on the outer devices because there is no need to transform any data outside the sources.

Any cutting-edge audio processing algorithms are resource-consuming, especially the speaker diarization that answers the question "Who spoke when?", so they can be utilized pretty easily on cloud servers like AWS or Microsoft Azure. However, what if we want to run these algorithms on smartphones that can't be as powerful as cloud services? The goal of this work is to make possible the utilization of the advanced speaker diarization models on low-resource devices. The execution of machine learning models on edge devices can be performed efficiently after optimization operations such as quantization or weight pruning which allows for speed up the execution and save the battery's charge. During this research, different optimization techniques will be explored and evaluated in their trade-offs between accuracy, speed, and model size.

In a century dominated by online communication over normal face-to-face conversation, Speaker Diarization [35] has risen as a crucial invention for transcribing oral communication. This method answers the question "who speaks when", supporting numerous modern applications, namely, meeting summarization, transcription, etc. Speaker Diarization closes the gap between human listening and machine listening, enabling more precise and contextually correct translation from spoken words to text.

1.1 Motivation

Speaker Diarization is clearly a promising technology as it solves different problems in communication like talking to a deaf person, but in practice, there exist several limitations with this technique. Firstly, the existing models which are based on Deep Learning are mainly focused on obtaining the best performance which leads to heavy computation and demand for a substantial amount of memory [3]. This leads

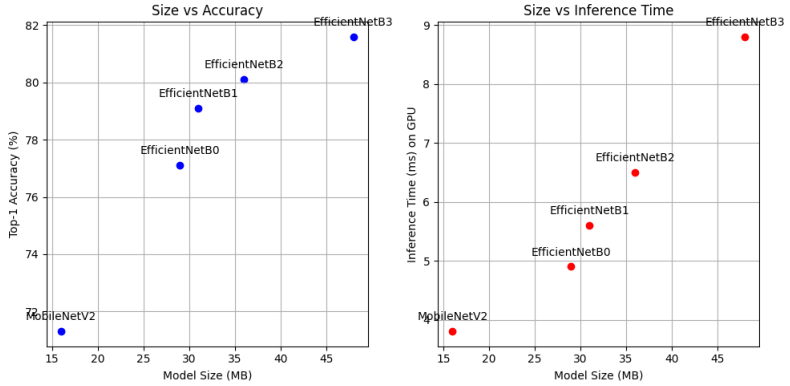

FIGURE 1.1: Performance of Keras models [25] on the ImageNet dataset

to problems in running the model on edge devices which usually are low-resources, namely smartphones or tablets.

For example, the Convolutional Neural Networks (CNNs) [33] that are popular for various applications, often rely on a high number of parameters to be able to achieve satisfactory performance. As we can see in Figure 1.1, representing different models [25] developed for the Image Classification task, the increasing number of parameters leads to better model performance. Such a correlation between the model's performance and computational demands leads to the potential infeasibility of utilizing cutting-edge models on-device for real-time applications. The Speaker Diarization models rely on combinations of various computationally demanding blocks, including Deep Learning models which makes it resource-demanding as well. A good example of advanced models that are unreasonable to run on edge devices is GPT models [3] which is too demanding, so all computations occur on servers.

1.2 Objective

The main goal of this research is to unlock the potential of Speaker Diarization in practice and prepare it to be executed on edge devices. We want to decrease the amount of resources needed to utilize speaker diarization in real time without hurting model performance. To realize this, we will focus on quantization and knowledge distillation. The quantization helps in lowering the precision of the model's parameters, therefore decreasing the model's demand for computing resources. This approach has been proven to substantially decrease the computational demand with no or small impact on the model's performance [24].

At the same time, the usage of knowledge distillation helps to create a smaller model with a smaller number of parameters which also helps to decrease the need for computing resources and to speed up the inference, ideally without harming the performance of the model [31].

The main focus would be two-sided: first, creating an optimized diarization model that is lightweight and fast enough to be run on the edge devices, and second, preserving the initial performance of the model or sacrificing only a small part

of it. Successful execution of such a plan will aid in the utilization of speaker diarization models within real-time applications on the different edge devices, meeting the demand for higher quality online communication.

Chapter 2

Context of the internship

2.1 What is Speaker Diarization?

Speaker Diarization is a process of predicting the labels of the true speakers to every segment of an audio stream. In short, we want to answer the question: who was speaking when[35]? There are many different architectures for this algorithm. For example, one of the most effective is when it consists of two parts: speaker segmentation and speaker clustering. The former is responsible for establishing the moments when the speaker changes and the latter has to group and assign different segments of audio to different speakers. Speaker Diarization can be divided into online and offline diarization, where online is a real-time form of technique and the offline version can use the whole signal for prediction. The practical usage of the algorithms is high-quality transcribing in real time.

2.2 Speaker Diarization in Industry

As we communicate on Google Meet or use Microsoft Teams we have already experienced some type of speaker diarization. In several services of these companies, speaker diarization plays a solid role. For instance, this technology is used in numerous applications, like transcription services, content indexing, and sophisticated speech analytics. In recent years, Microsoft has created speaker diarization services for several applications, like Azure Cognitive Services. Also, Azure Cognitive Services uses Speaker Recognition API that allows the developers to utilize speaker diarization algorithms within dedicated services.

On the other hand, Google has a video service what is called Google Meet. During the call, one can select transcribing which later yields a text file where each speaker will have their text. Such technology helps to capture different speakers on the video, so it can create a new version of subtitles with the assigned identities, instead of a plain transcription.

2.3 Company

Another company that actively utilizes the capabilities of speaker diarization is Ava which develops and uses a modern speaker diarization model. Intended as a diarization service that works in real-time, the model uses modern AI methods, either during video calls or normal conversation, producing speech transcripts colored with the corresponding speaker tags.

Ava was founded in 2014 with a mission to enable 466 million deaf and hard-of-hearing people to communicate without barriers. Ava built a team of over 50 people from all over the world. The main engine of the company is the AI team

FIGURE 2.1: AVA application

that works on different audio applications of machine learning. The technology that Ava is building helps people overcome communication barriers and work together to achieve their goals.

Within our digital era, one of the main questions is how to communicate more and more effectively. In this sense, the factor of inclusivity can not be neglected as hearing-impaired people should be involved in conversations fully, so the main goal of Ava is to empower deaf and hard-of-hearing people with cutting-edge applications that break the communication barriers isolating them from other people. The company helps not only with business or working situations but also with education and even normal conversations. The application is integrated into several schools to help deaf students access oral communication but also can be used to connect people during lunch breaks.

2.3.1 Company's Product

Suppose you are in a team meeting with several speakers. Then, with the help of the Ava application with speaker diarization model 2.1, which was trained with the usage of a dataset gathered by Ava, the audio can be segmented into different speakers in real-time with higher accuracy than other similar services. Such an opportunity is an advantage for hearing-impaired people, also it helps with the review of the meeting without the need to replay.

The Ava application boasts one of the most advanced and fast real-time transcription features. By combining several advanced algorithms and scientific developments in speech recognition and natural language processing, the service effortlessly transcribes spoken words into text in real time. With this feature, people can easily follow the flow of conversations and improve their analysis of the recording as they don't have to listen to the recording again. The application can also synchronize with different devices at the same time, so the conversation can be accessible from different places around the world at the same time.

Chapter 3

Related work

3.1 Speaker Diarization

An advanced audio processing method which we call speaker diarization is the same as writing a diary of events in the conversation. Speaker diarization consists of separating the audio with several speakers into different segments that belong to different speakers. In short, the process can be described as answering the question of "who spoke when". In a nutshell, it is a sequence of several algorithms each of them doing a separate task: speech/voice activity detection, speaker overlap detection, speaker recognition, and speaker count estimation. Thus, speaker diarization doesn't need any information about speakers during its implementation, the number of speakers will be identified during the execution.

The traditional way of addressing speaker diarization is to divide it into different blocks as can be observed in Figure 3.1 [35]. Firstly, speech techniques like front-end processing solve some acoustic issues and perform feature extraction. Followed by speech activity detection or SAD for short. During this step, the audio is separated into segments with or without speech. Then, the embeddings are extracted from the segments.

Finally, clustering is the process where the model identifies which chunk of audio belongs to which speaker [10]. These results can be modified by post-processing algorithms in order to increase the accuracy of identification [26].

However, in this work, we will focus on an overlap-aware low-latency online speaker diarization model based on end-to-end local segmentation [8]. This type of model was developed as one single neural network. This network performs all tasks as one without the need for different submodules to post-process or cluster segments. As the model heavily relies on the End-To-END diarization model [13] will focus on its principles.

End-to-end neural diarization or shortly EEND at the first time of publishing was a ground-breaking model for recognizing and separating speaker identities during the conversation. The main idea in the presented approach [13] was to redefine speaker diarization into a multi-label classification task. Such change was motivated by the idea that the standard diarization system's steps which are separated into speech activity detection, speaker segmentation, and clustering, can be merged as

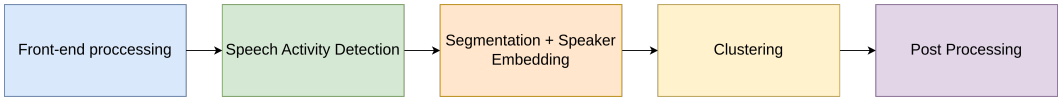

FIGURE 3.1: Standard speaker diarization system

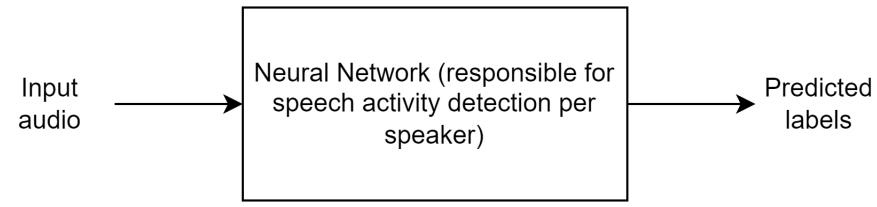

FIGURE 3.2: EEND speaker diarization system

we can see in Figure 3.2 into one operation which would be more efficient. The main feature of this model is the simplicity of the defined task and the identification of the speakers in a multi-label classification manner.

The objective of the approach is to represent speakers during the course of the recorded conversation in frame-wise speaker activity probabilities, which are later binarized.

One of the main innovations of the EEND approach is the usage of permutation-invariant training [30]. The presented loss function [19] encompasses all conceivable combinations of possible speaker labels since the order of speakers in the binary frames is random. Consequently, the trained model is capable of making predictions without prior knowledge of the speakers' order. However, it's important to clarify that the challenge lies in a matter of aligning speakers in the prediction with speakers in the reference, which is done over the most probable alignment, based on the binary cross-entropy loss To be precise, EEND utilized on-the-fly audio augmentation [34] which increases the quality of extracted features during the future inference process as the model becomes more robust to the noise.

To summarize, the model that we will utilize in this work is significantly different from the traditional approach. Using permutation-invariant training and high temporal resolution, approaching the speaker diarization task as a multi-label classification task the over task became easier. Such an approach is a solid idea in the audio analysis field and has the potential to be used in a huge number of different applications where speaker diarization is crucial. One of the main advantages of such architecture is a feature to handle the overlap speech which can be really useful for online speaker diarization. For example, during the video call people tend to speak simultaneously, so such feature would be helpful.

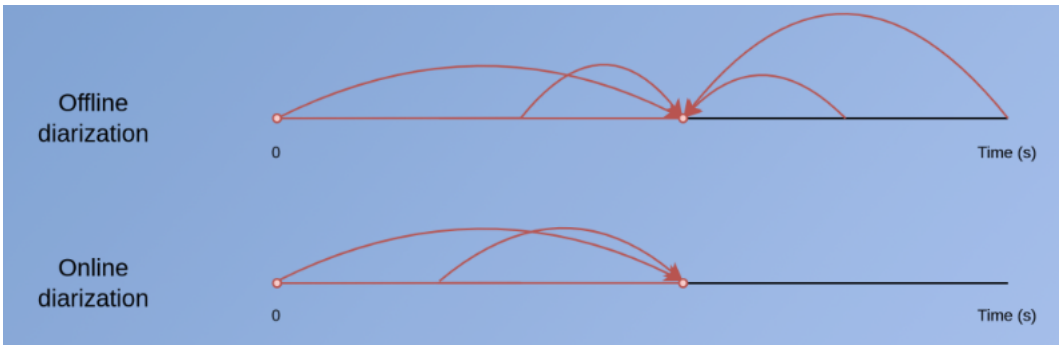

FIGURE 3.3: Comparison of information usage in online vs offline diarization

3.1.1 Online vs Offline Diarization

The speaker diarization domain can be divided into two different domains: offline and online speaker diarization. The difference between them is observable in Figure 3.3 can be drawn from their names: offline speaker diarization operates with entire audio files and can use the information from the end of the file to predict the speakers at the beginning of the file, while online diarization has to operate in a real-time manner which doesn't allow much time for computation and especially utilization of the information obtained from the end of the file in the start of it. In the following paragraphs, we dive deeper into these concepts to better understand their differences, and why we choose to focus on the online case

Offline Speaker Diarization

Answering the question "who spoke when" for prerecorded audio is called offline speaker diarization. In opposition to real-time models, offline speaker diarization does not constrain information available which allows for the utilization of more resources, time, and training data, so the analysis is full and thorough.

This process is split into different tasks in the standard offline speaker diarization approach [21]. We start with the already discussed algorithm: SAD which is the preliminary step that is responsible for identifying the chunks of the audio where sound is present. After identification, these parts are examined with our next algorithm. From these speech segments, the speaker embeddings are extracted which are representations of the acoustic features. From these features, the unique qualities of each speaker can be extracted. For the next algorithm which is clustering these embeddings are used as inputs. Extracted embeddings are grouped with the utilization of clustering algorithms the main role of them is to divide the audio into distinct speaker chunks. It is finished with methods that help to deal with overlapped speech which is handled by specially designed overlap handling algorithms.

After the creation of the standard cascaded approach several improvements were created [9, 44, 16]. At first, some research was focused on the improvement of extraction of the embeddings. The new ones should be more accurate in capturing speaker subtleties. Also, clustering algorithms were improved in terms of accuracy and efficiency. And finally, the overlap handling methods became more and more sophisticated which resulted in better diarization results. However, as we can see the improvements of the standard approach can be completed only in a cascade manner. The other way of changing is to interact with architecture.

The main advances in the change of the architecture are end-to-end approaches [16] which are now mainstream. As they don't contain any specific clustering and segmentation steps, such models have trouble have problems with handling the different numbers of speakers. However, in combination with different clustering methods, the end-to-end approach yields better results. The advantage of such a hybrid is that an unidentified number of speakers can be used thanks to the presence of clustering methods, so the simplicity and interoperability of end-to-end methods remain.

To summarize, offline speaker diarization utilizes prerecorded audio for the separation of the distinct speaker segments. The traditional approach is conducted with a cascade of speech activity detection, embedding extraction, clustering, and overlap handling. Some improvements were made in this field with improvement separately of each part of the algorithm. Another possible way is to change the architecture and use one model based on an end-to-end approach which can be also upgraded with the addition of a clustering method at the end of the chain.

Online Speaker Diarization

In contrast to offline diarization which utilizes previously recorded audio, online speaker diarization is a method that identifies and identifies speaker utterances in real-time. This type of application can be useful for a wide range of tasks like online calls, broadcasts, or any other setup where real-time transcribing is necessary. Following its offline counterpart, online diarization can also be implemented in a cascade or end-to-end.

When we are talking about the cascade approach the diarization pipeline's elements must perform in an online manner which puts them at a disadvantage compared to the offline diarization where each part has much more time and resources to operate. For example, for offline clustering the popular method VBx [39] has a normal performance, however, if one tries to implement it for the online setup it will struggle [22] as it depends on two-stage clustering for refinement. One of the crucial parts of this process is the clustering of speaker embeddings. Online clustering usually yields worse results the the offline one, because of the time and resource constraints. The standard clustering approaches don't work as they are effective after obtaining a huge amount of data, so for this situation, different algorithms are utilized, namely UIS-RNN [43], UIS-RNN-SML [42], constraint incremental clustering [47], etc., they adapt to changing speaker identities in a streaming audio feed by iteratively updating speaker embeddings and calculating speaker numbers.

As mentioned before, end-to-end approaches can also be applied to online speaker diarization. This direction of research is focused on developing machine learning models that can process the audio segment without a huge amount of resources and output the speakers' labels. In order to prepare the model for real-time usage one method is to train them in a frame-wise or block-wise manner, this is also a limitation of such models. They can't be used for offline diarization as they can't ingest entire conversations. Popular models that utilize such ideas are BW-EDA-EEND [5], and Online RSAN [6]. The other example, the speaker-tracing buffer method [29] utilizes the EEND model in an online manner and caches the acoustic features and the diarization output of the previous frame to solve the speaker order problem. Such results can be more than promising and show similar results to the models that are designed from scratch.

To summarize, training the model for the online speaker diarization is sometimes challenging due to the lack of ground truth labels, the streaming nature of the data,

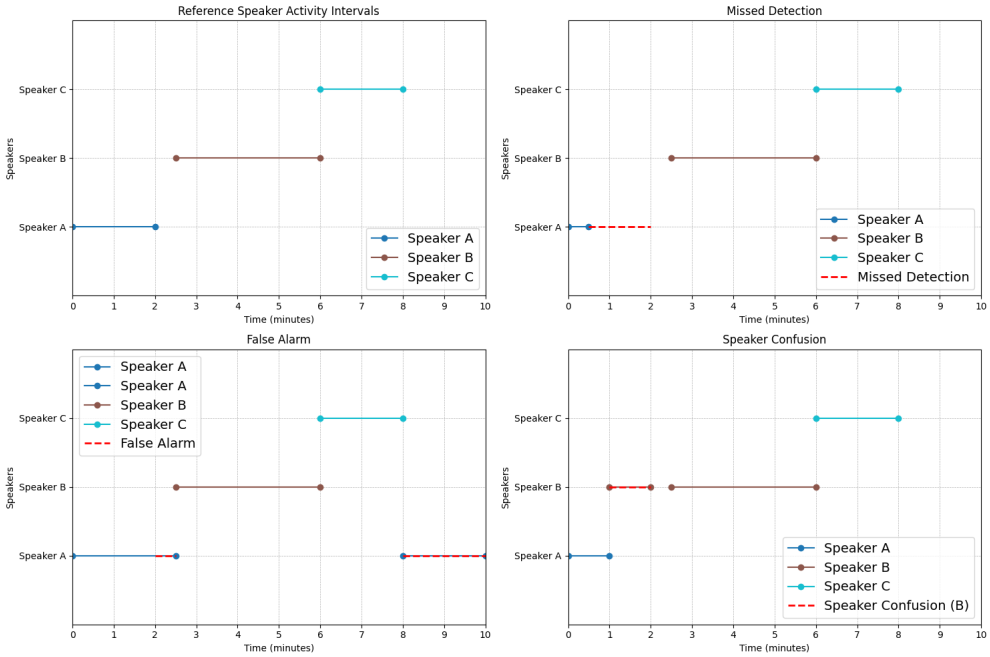

FIGURE 3.4: (a) Top left graph shows the actual reference of speakers' activity (b) Top right graph shows prediction where Missed Speech error showcased and outlined as red dashed line (c) Bottom left for False Alarm error (d) Bottom right for Speaker Confusion error

and the requirement to adjust to fluctuating speaker characteristics, training online speaker diarization models is difficult. As we previously discussed, the main focus of this research is to optimize the speaker diarization models with a particular focus on their online and real-time usage. It is worth mentioning that it is possible that several voices are present in the audio at the same time, so several classes can be selected at the same time. Fsinc

3.1.2 Evaluation criterions

The evaluation of the speaker diarization task is based on the metric which is called Diarization Error Rate [12]. It consists of three main parts which are errors in speaker boundaries, speaker identities, and false alarms. For comparison of different diarization methods and their evaluation of effectiveness, the DER is used as the main criterion. Smaller errors in speaker identities, boundaries, and duration are captured by lower DER, which shows better performance.

$$\text{DER} = \frac{\text{False Alarm (FA)+Missed Speech (Miss)+Speaker Confusion (Conf)}}{\text{Total Duration of Speech"}}$$

Missed Detection

Missed Detection [12] or short Miss is an error caused by speech segments that are on the reference annotation but were missed by the speaker diarization model. An example of this error is shown in Figure 3.4.

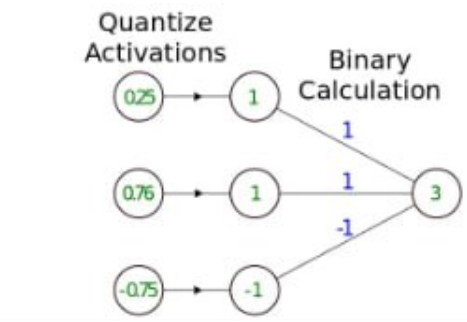

FIGURE 3.5: Quantization of Neural Network with Z=0 and S=1

False Alarm

False Alarm [12] (or FA for short) occurs when the model incorrectly detects the presence of a speech segment while it is not present on the reference annotation. This error can be caused by different factors such as non-speech sounds like noise. An example of this type of error is shown in Figure 3.4.

Speaker Confusion

Speaker Confusion [12] or short Conf is an error in identifying the correct speaker. Such mistakes are caused by speaker overlap and similarity between different speakers. An example of speaker confusion is shown in Figure 3.4.

3.2 Quantization

Quantization [23] is a fundamental concept in signal processing and data compression that involves reducing the precision or granularity of numerical data. In this process, continuous values are approximated or represented by a finite set of discrete values. The goal of quantization is to strike a balance between preserving the essential information of the original data and reducing the amount of storage or transmission capacity required.

In quantization, a continuous range of values is divided into a finite number of levels or intervals. Each level is represented by a discrete value, typically represented by binary digits (bits) in digital systems. The quantization process involves mapping each input value to the closest discrete level. However, due to the finite number of levels, quantization introduces a form of distortion known as quantization error. This error arises from the difference between the original continuous value and its quantized representation [46]. The level of distortion depends on the number of quantization levels and the distribution of the input values. By adjusting the number of levels, the trade-off between the level of distortion and the required storage or transmission capacity can be controlled.

The quantization of neural networks works as compression of the weights of each layer as we can see in Figure 3.5. The basic quantization formula is

$$Q(r) = Int(\tfrac{r}{S}) \text{ - } Z$$

where r is the real-valued input which can be either activation or weight, S is the scaling factor which is a real value, and Z is an integer zero point. Suppose the values that we want to quantize lie in the range of -1.5 to 1.5. This type of quantization

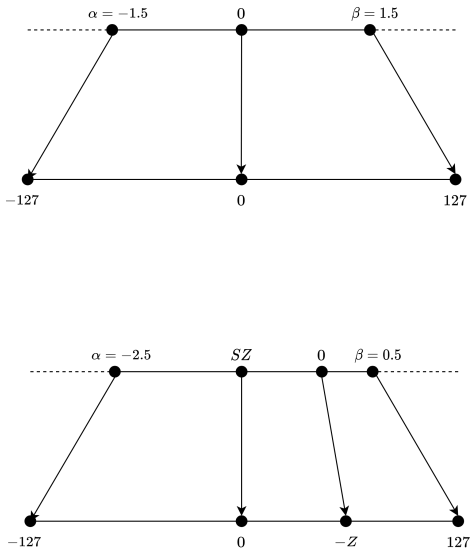

FIGURE 3.6: 8-bit quantization with ranges of -127 to 127. On top is a symmetric quantization with a clipping range [-1.5, 1.5], and on the bottom asymmetric quantization with a clipping range of [-2.5, 0.5]

is called uniform which can be seen in Figure 3.6 as the quantized and real values are uniformly transformed. The quantization also can be non-uniform as we can see in Figure 3.6, where the quantized values are not uniformly spaced as we can see that 0 doesn't transform to zero but the $S * Z$ does.

If one wants to go from a quantized number to an initial format, then dequantization can be used. The dequantization operation can be described as

$$\hat{r} = S(Q(r) + Z)$$

However, the dequantized value usually is not equal to the initial one, because of rounding errors.

The important differentiation factor is whether the quantization is symmetric or asymmetric. As we have seen from the equation 3.2 the scaling factor S has a significant influence on the result of quantization. This factor divides the input range of values into a number of partitions:

$$S = (\frac{\beta - \alpha}{2^b - 1})$$

where the clipping range is defined as $[\alpha, \beta]$ which is the range that we bound the input real values with and b is the precision of quantization. Therefore, the choice of the clipping range is the first essential step of the quantization process [11]. Suppose we have a quantization function Q and the set of numbers that will be quantized with this function will be referred to as R. How do we choose a clipping range based on the set of numbers from R? The straightforward approach is to select the maximum as β and the minimum as α. If $|\alpha|$ doesn't equal $|\beta|$ then we would get a situation with asymmetric quantization. However, the most popular approach and common for most frameworks is to select $\alpha = -\beta$, where β is equal to the maximum absolute value of R. Logically, such an approach is called symmetrical. Despite its popularity, there are certain situations where the asymmetrical approach is more effective. For example, during the work with different activation functions like ReLU,

the clipping range would be half the size in cases of asymmetrical quantization as there won't be numbers up to zero.

The other approach to the calculation of quantization variables is to take into account calculating the clipping range only some percentage of values that go through the quantization function. The previous approach is vulnerable to outliers that can influence the whole process and decrease the accuracy of the post-quantization model [40]. The first solution is to simply select the i-th percentile for minimum and maximum values. More advanced methods suggest using the KL divergence and minimizing it between original and quantized values.

There are many strategies to calculate the clipping range for the quantization function. However, in order to evaluate this range there is a need to select the values on which such calculation will be based whether is obtained during the training, inference, or some special moment. Such a question is even more important in terms of the weights of neural networks, whether we rely on two types of quantization methods, dynamic which quantizes on-the-fly during inference, and static, which quantizes once and for all without any additional processing during inference.

3.2.1 Dynamic Quantization

The first type of quantization utilized in this work which is one of the cornerstone techniques is dynamic quantization [14]. With the usage of this method, the activation and weight of the neural network are represented as integers. The core difference between regular quantization and dynamic is that the dynamic one is applied after training which leads to a faster model.

The evaluation of the scale and zero-point is done during the inference phase using the data obtained during this inference which is the main concept of dynamic quantization. Such an approach doesn't harm the performance(or by a small margin) [14] and allows more accurate weight representation during the quantization.

The first advantage of such an approach is the lower memory footprint of the model [24] as we replace the higher-precision floating-point numbers with low-precision integers. This property favors the system especially edge devices as they dispose of small computational resources, so such an advantage increases the usability of the quantized model of this type of device.

Also, model execution is sped up by dynamic quantization as the computational cost is reduced as the integer operations are less demanding in terms of resources as well as time, so the latency of the quantized model will be decreased. This feature will be helpful for an application that relies on real-time interactions where low inference time is essential. It is worth mentioning that despite a decrease in inference time due to transformation for integers the dynamic quantization process occurs during the inference time which makes this time particular quantization type slower compared to other quantization types.

The final advantage of dynamic quantization is the fact that it is applied to the model after the training, so the initial model can be retained or fine-tuned. Therefore, great flexibility is offered by this type of quantization as the improvement of the model does not necessarily have to be performed from the beginning. For example, if the model in production needs to be fed with a newly obtained dataset which is essential for a higher performance the model can be dequantized (if the initial model is not saved), retrained, quantized again, and put in production. It is worth mentioning that this approach doesn't need to be coded by one as frameworks like PyTorch [37] has incorporated this particular type of quantization, so almost anybody can use

it to their advantage and obtain a model with less memory usage, faster execution and with the possibility of improving the model.

To summarize, the benefits of dynamic quantization are valuable for the production usage of the models based on neural networks. The model requires less memory to run, the inference time will be lower and the model can be refined without huge effort. Dynamic quantization preserves as much information as possible for such type of optimization as the evaluation of the clipping range is done during the inference time, however, this slows the inference time, so the other approach was discovered to handle this disadvantage.

3.2.2 Static Quantization

Contrary to the dynamic approach, in static quantization to make an inference of the Deep Learning models fast and less resources demanding is a quantization method called static quantization [14]. It also converts the model's weights and activation into low-precision integers, typically 8-bit. However, the computation of clipping range, scale, and zero-point is done before the actual inference, so it saves time compared to the dynamic one. In a situation where computation resources are rather limited, which is typical for smartphones or other edge devices, or even for computers when there is a need for fast computation, static quantization is a good solution as it decreases memory footprint without a noticeable decrease in performance in most situations.

Static quantization has a different algorithm compared to dynamic and can be divided into two separate steps: calibration and quantization [24] itself as can be seen in Figure 3.7. As we don't want to collect the data about weight and use them for calculation of clipping range we try to collect the information about weights and activations before actual inference. A special dataset is used for acquiring the statistics about the model's parameters, especially minimum and maximum, so then they will be used to determine all important parameters for the quantization like scale, clipping range, etc. In order to perform quantization as accurately as possible the special dataset should be a good representation of actual data that will be inputted to the model during the inference.

After the calibration, the actual quantization happens. With the statistics collected during calibration, the model's parameters float values are mapped to their integer quantized versions. This transformation is based on the collected statistical values like minimum and maximum, outliers, etc. After this step, we obtain a model that operates in lower precision, typically 8-bit and is ready for actual inference.

From a usability point of view, the models don't need to be retrained in standard static quantization variation, so the models which are usually trained on usual float values, don't need any additional transformation, except quantization itself. Such features allow to use of static quantization with pre-trained models without any additional prework.

Static quantization is a method that offers a number of advantages to the scientist which are up to utilize deep learning networks. The first one, the same as for the dynamic counterpart, decreased inference time and lower memory bandwidth which is a result of lower weight precision. The performance can be additionally ameliorated if the framework and hardware support integer operations. The second one is a reduction in memory footprint, as we conduct all calculations is lower precision the model has less memory footprint than the original one, adding another advantage for the models that have to be used on low-resource platforms. And last

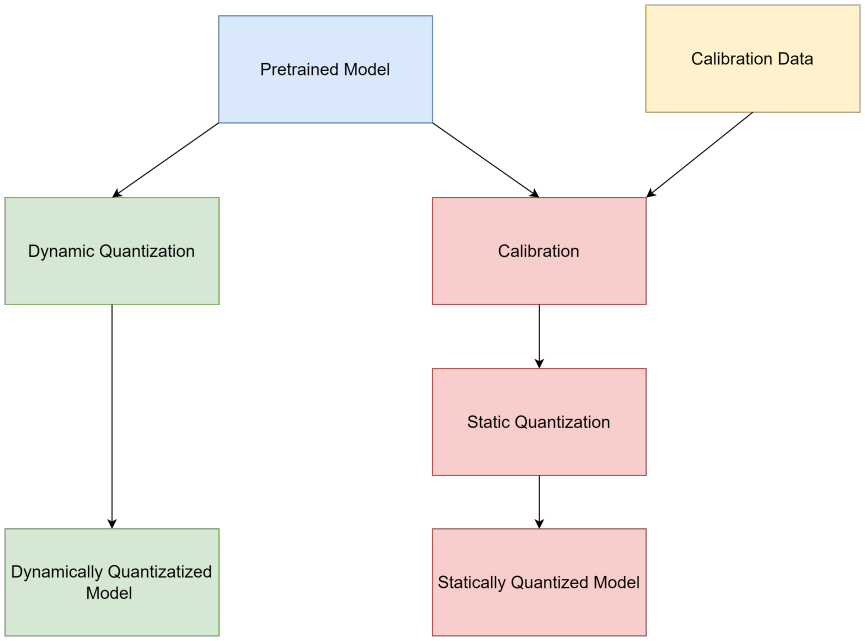

FIGURE 3.7: Algorithms of dynamic static quantization

but not least, is more effective energy usage as the integer computations spend less energy compared to the floating point operations.

Static quantization offers many valuable advantages, though the disadvantages of such a method can be too important in some setups. If we compare with the unquantized model, the accuracy can be noticeably decreased in some cases. Many factors influence the performance of the quantized model [48]. Firstly, its architecture can rely on functions or weights that can have high variance. This results in inaccurate transformation, subsequently resulting in worse performance. Also, it can be determined what loss of accuracy is acceptable for the quantization and work with these numbers. The second problem occurs with the calibration dataset as it has to be selected carefully. During the inference, all weight statistics should be representative and show the actual distribution of the activations, so the calculated clipping range and other variables will be accurate and correctly address all possible values during inference. Therefore, the calibration set should be selected with great attention to the diversity of the dataset. Last but not least, many different hardware nowadays are compatible with integer operations, which is necessary for static quantization. However, they should also be compatible with the static quantization itself as well as with the framework on which this quantization is realized. For example, several methods and types of layers can be used in PyTorch framework [36].

To summarize, static quantization is a powerful technique for improving a neural network's efficiency in terms of faster inference, less memory requirements, and better energy usage. In the case of careful and calibration dataset selection, the obtained model will be optimized without a huge loss in accuracy [48]. Nevertheless, in a situation where the loss in accuracy is inevitable, we should consider the trade-off between accuracy and inference time and memory footprint with great attention and determine the limits of DER growth for usage in actual applications.

3.2.3 Quantization Aware Training

The last quantization method is Quantization Aware Training (QAT) [14]. Static quantization can decrease the accuracy of the unquantized model with a reduction in the need for resources and memory footprint, these Neural Networks that were optimized with static quantization can regain their accuracy with another optimization technique, QAT.

To do this, QAT trains the model in a quantization-aware manner, aiming to improve the quality statistics for the clipping range of static quantization. In traditional training schemes, during forward and backward passes floating point values are used. However, in QAT both these passes simulate the behavior of the quantized weights and activations. During the passes, because of lower precision quantization error occurs which are differences between original weights and weights with simulated quantization. Therefore, such errors are taken into account and the model becomes less vulnerable to the negative consequences, namely loss of performance caused by bad quality of precomputed statistics, of possible quantization, and its accuracy is preserved.

The essence of the algorithm is in operations with fake-quantized weights [28]. The training is done with floating point values, but the weights and activation are quantized during each weight update with a gradient, but the gradient itself is a floating point value. Such an approach prevents the problem of vanishing gradient as the computation of gradients in the integer values will result in a gradient equal to zero which results in no updates to the weights and subsequently to high error [28].

The advantages of the QAT are the same as for static quantization, it requires a smaller amount of memory and energy, which makes the model more compatible with edge devices. However, due to the special training, the loss in accuracy is decreased compared to the classic static quantization.

In summary, QAT is an optimization technique that improves static quantization and partly negates the loss of performance of the algorithm. At the end of the day, we get the optimized model which is faster, more resource-efficient, and has higher accuracy than a regular static quantization, but needs more data than static quantization and can still suffer from loss of performance.

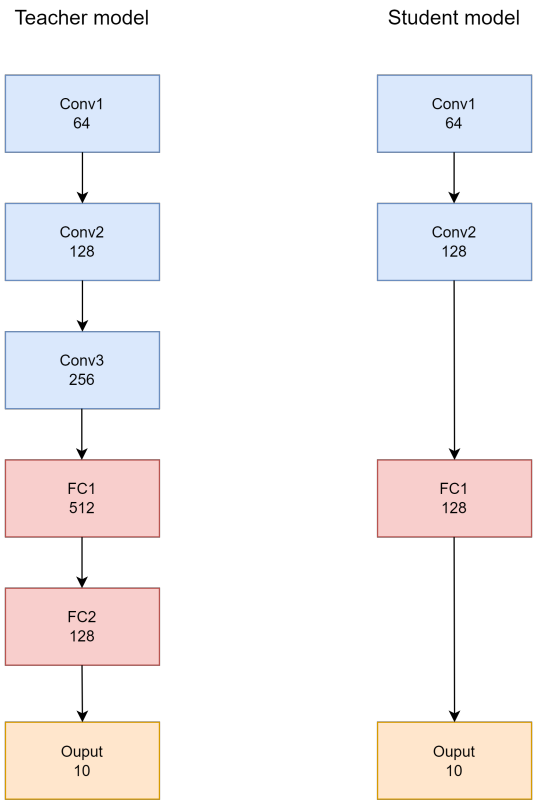

FIGURE 3.8: Example of Knowledge Distillation in Convolutional Neural Networks, where Conv1 64 is Convolutional layer with 64 filters and FC1 128 is Fully Connected layers with 128 neurons

3.3 Knowledge distillation

Knowledge Distillation [20] is a machine-learning technique that allows transferring the knowledge from the teacher model which usually has a large number of parameters and has good performance, and the knowledge to the student model which is usually a compressed version of the teacher model that needs. In short, knowledge distillation is a process of training the smaller model from the bigger one using the predictions of the latter.

As a technique, Knowledge Distillation is used in the Deep Learning field which allows us to transfer knowledge from large-scale, sophisticated, and complex models into smaller but more efficient models for a particular task. During this operation, the partnership between two models which are referred to as teacher and student work. The teacher model like in Figure 3.8 which was created originally as the model with more parameters, and more complex architecture that has a good performance but is too large or slow for a particular task. The student model has a smaller number of parameters but its task is to obtain the teacher's knowledge, so architecture should be at least capable of regaining them.

In order to train the student model with the teacher model's knowledge we use soft targets. The idea of soft targets is one of the most important concepts. It ensures that not only simple labels are provided to the student model, but also additional information about the certainty of each label by probabilities at each label. Therefore, for training our models, we don't utilize hard targets but soft targets. Such targets are the probability distribution of each label for all classes. These targets give the student more significant information about the relation between different classes and help to generalize better. The calculation of soft targets can be presented as:

$$\text{Softmax}(z_i) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \tag{3.1}$$

Here, each z_i represents the target score for each class before activation functions for a specific class and T is a temperature, which is responsible for the sharpness of the targets

Soft targets are created with the help of a teacher model as the raw class scores are taken and run through the softmax function with the usage of particular temperature parameters. The generated predictions through this algorithm are "soft" as they show the teacher's probability prediction for each label, so the student will generalize better. The hyperparameter that is used in such transformation is called softmax temperature and it helps to control the sharpness or smoothness of the probabilities. If we increase the softmax temperature the final distribution of probabilities will be more even across all labels. If we want to highlight the teacher's certainty in some classes, we would use lower temperatures for this purpose.

The main goal of the student is to predict the teacher's soft targets during the training, the student shouldn't match the exact labels of the teacher's prediction to avoid overfitting, but rather generalize based on them. Such an approach opens the possibility for students to acquire knowledge from the teacher in all possible situations, even when a teacher doesn't give a strong prediction in some cases. Another factor for successful training is to fine-tune the softmax temperature. In order to balance the student's capabilities of generalization and the teacher's certainty in labels the correct temperature should be selected.

For training student models in the knowledge distillation setup, the selection of good and representative loss functions is essential [15]. The most popular choice in such situations is Kullback-Leibler divergence loss or KL for short [18]. This function rather than a direct comparison of student predictions with labels, evaluates the difference between soft probabilities presented by the teacher, and the student's predicted class probabilities. Such properties don't force the student to give a hard prediction but rather move into the teacher's direction of prediction. The other popular loss is cross-entropy loss, which also forces the student to match the teacher's prediction, but rather just replicate the probabilities of predictions. In order to determine which loss function should be used in different applications the experimentations are necessary.

As we already discussed one important factor during training is the softmax temperature parameter. Temperature scaling is a technique that changes the temperature during the training. When the selected temperature during the training is 2 or higher as we can see in the equation 3.1the student is induced to find more solutions which can lead to better generalization. Contrarily, a temperature lower than 1 makes the difference between soft targets more visible which forces the student to more closely resemble the teacher's predictions because they haven't been disturbed

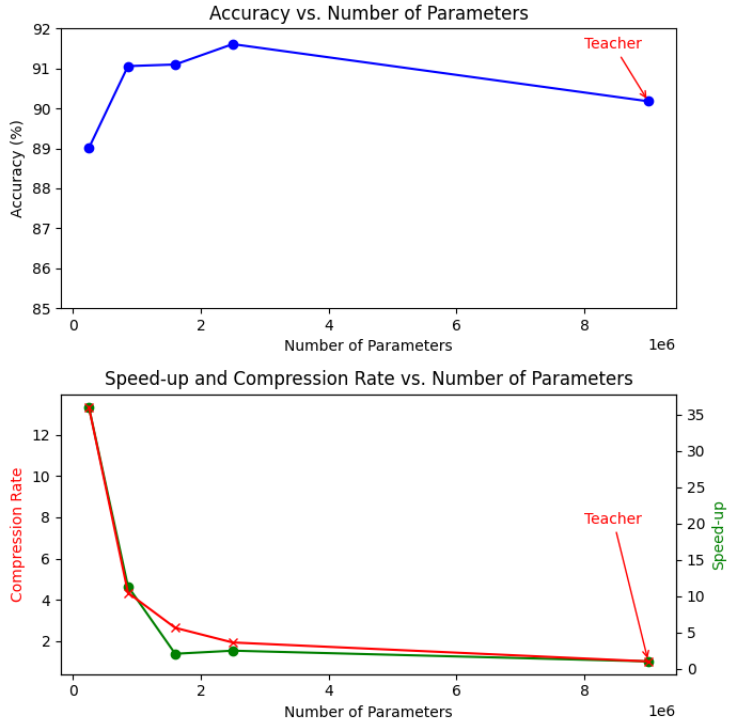

FIGURE 3.9: Accuracy, Compression, Speed/Number of parameters trade-off on CIFAR-10 in [41]

by the temperature. The selection of the correct temperature is based on the particular dataset and the task of the model. The lower temperature may lead to overfitting, while the higher temperature can make the student inaccurate, so the selection of the right temperature is essential. It can be presented as a fine-tuning of a hyperparameter, so the different parameters will be selected and then tested for the particular experiment. It can be also adjusted during the training if there is a need to force the student into different directions.

There are different techniques that can be utilized during knowledge distillation. One of them is to distill the internal weights of the model and try to ensure that the student model not only tries to replicate the output of the teacher model but also copies its internal parameters to some extent [17]. Such an approach helps students to learn not only the label probabilities but also the internal details that led to such conclusions [49]. Another way to ensure that student learns some key information is the transfer of attentional mechanism. In this case, knowledge distillation can include the teacher's attentional mechanism to pass to the student in order to push the student's focus on special areas of the input data. Another approach to make the student better generalize is to use different teacher models in an ensemble way [50]. Due to two or more teachers, the student's ability to generalize will be improved and it will obtain different knowledge from different teachers. The student can also gain knowledge from itself. This method is called self-distillation [27] and during it, the student tries to match with its own predictions. Finally, the distillation can be done not only between the models with similar architecture but with a different one and with a smaller number of parameters [1]. The architecture can be changed completely, so knowledge will be transferred from complex architecture to a simpler one. The main question would be is the student capable of repeating such processes.

3.3.1 Properties of Knowledge distillation

Knowledge distillation is an effective optimization technique that has many benefits and makes models more efficient. It has its limitations, but the advantages are more than valuable if the goal is to decrease the inference time and make the model less resource-demanding.

The usage of knowledge distillation provides several advantages. The first one is the reduction of the model size. As we can see in Figure 3.9 the reduction in the number of parameters doesn't worsen performance at all except in extreme cases when the number of parameters decreases too much. It is also accompanied by a reduction in memory requirements, so it increases the usability of the model on resource-limited systems. It also helps to decrease inference time as can be seen in Figure 3.9 as the model is decreased and the number of computations is decreased so is inference time. Knowledge distillation also helps with generalization beyond training data due to training with soft target and management of the softmax temperature which yields similar to the original model or even better performance on unseen data [17]. Last but not least teacher can be used without the dataset on which it was trained. Therefore, the sensitive information that the owner of the teacher model doesn't want to share will be included in the training without breaking its privacy directly as some information can be obtained only from the model's weights.

However, there are several disadvantages to the proposed algorithm. Firstly, in specific cases despite all modeling efforts some information can be lost during the training which can result in worse performance, especially when the student model is much smaller. Next, the performance of the student is based on the teacher, so the student will probably inherit the mistakes of the teacher. The training of the student is complex due to the selection of hyperparameters which can result in long experimentations. Also, the risk of confirmation bias is high, so without proper attention to regularization, the student can simply copy the teacher's answers.

To summarize, knowledge distillation is a powerful optimization method that helps to create a smaller, faster model, and with the same or even better performance. It can be executed in different manners and even several teachers can be used to ensure proper training. This technique would be helpful for the development of well-optimized models, so they are less resource-demanding and can be deployed on edge devices.

Chapter 4

Experiments

As it was declared the main objective of this research is to decrease model size and inference without loss in performance of the speaker diarization model that is used in speech recognition. The main focus will be on evaluation metrics such as DER, inference time, and model size. During experiments, we utilize the following methods: quantization, knowledge distillation, and a combination of them.

4.1 Model description

The model we optimize is the speaker diarization model 4.1from the pyannote library, similar model to which is currently used in production in several applications at Ava. From Figure 4.2 we can see that the model consists of five layers: SincNet, LSTM, Linear, classifier, and activation.

Sincnet [38] layer is an important feature extractor for audio streams. It is a type of CNN layer and is designed to capture speaker-specific information from audio signals. Next are 4 LSTM layers which are bidirectional, so the LSTM can capture contextual information from both past and future time steps because it processes input sequences in both forward and backward directions. They process the extracted feature from Sincnet and capture sequential patterns and temporal dependencies. After that comes linear layers which process extracted features by LSTM layers. Then there is the classifier layer which is the linear layer that maps obtained features into scores of classes (in our case there are 3, but the number can be selected separately), so everything is done in an end-to-end fashion. And finally comes the activation layer which is the sigmoid that transforms the score into probabilities of each class speaker.

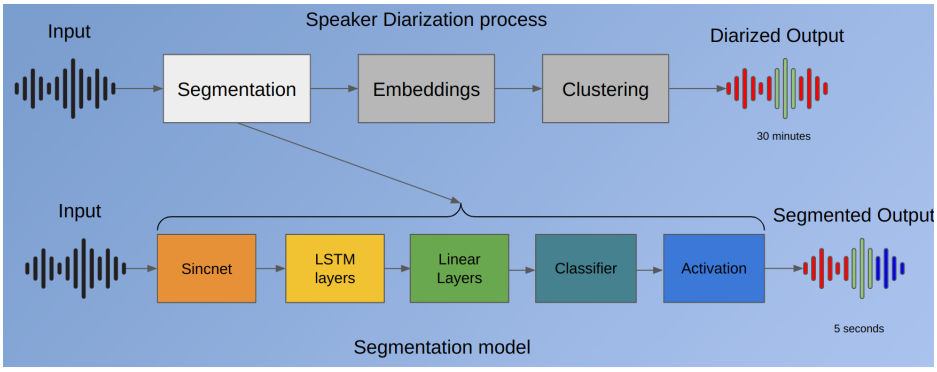

FIGURE 4.1: Architecture of the speaker diarization and segmentation model

```
  | Name       | Type       | Params | In sizes      | Out sizes
------------------------------------------------------------------------------------------------------------
0 | sincnet    | SincNet    | 42.6 K | [3, 1, 32000] | [3, 60, 115]
1 | lstm       | LSTM       | 1.4 M  | [3, 115, 60]  | [[3, 115, 256], [[8, 3, 128], [8, 3, 128]]]
2 | linear     | ModuleList | 49.4 K | ?             | ?
3 | classifier | Linear     | 387    | [3, 115, 128] | [3, 115, 3]
4 | activation | Sigmoid    | 0      | [3, 115, 3]   | [3, 115, 3]
------------------------------------------------------------------------------------------------------------
1.5 M     Trainable params
0         Non-trainable params
1.5 M     Total params
5.891     Total estimated model params size (MB)
```

FIGURE 4.2: Architecture of the PyanNote segmentation model

4.2 Dataset

TABLE 4.1: AMI-SDM Dataset Statistics

Dataset Split	Hours	Recordings	Speakers
Train	81	98 meetings	25
Dev	9	20 meetings	5
Test	9	20 meetings	5

The dataset that was used during training is AMI-SDM [4] which is a popular tool for research in audio processing, speech recognition, and speaker diarization. This dataset is a Single Distant Microphone (SDM) recordings subset of a larger corpus of Augmented Multiparty Interaction or AMI. It proposes an archive of meeting recordings including audio, and video.

The dataset consists of recordings in English in the amount of around 100 hours as we can see on Table **??**. It is worth mentioning that most of the participants are not native English speakers [2]. Also, recordings were done in many different spaces with various acoustic environments which ensured that numerous scenarios were included. Each participant usually speaks turn by turn, but sometimes their speech intersects with each other. Such properties are helpful in the audio analysis and evaluation of speaker diarization models as the data is properly processed and correctly labeled, so there is no need to do it by yourself.

4.3 Optimization by quantization

Our first optimization method is quantization. The main goal of this part is to test all possible methods of quantization and obtain a smaller, faster model without loss in performance.

4.3.1 Methods

Three different methods were used during this stage. The first one is dynamic quantization. It is applied to pre-trained models and the main characteristic of this method is that the quantization parameters are calculated during inference which results in a lower increase in DER, but higher inference time compared to other quantization methods.

The next one is static quantization which is also used for pre-trained models but requires a calibration step. During this step, the necessary statistics will be collected for each quantized weight, and then based on it quantization will be performed. This approach greatly reduces inference time and model size but can increase the

DER of the model. It is also a framework and system-demanding method, so not every machine and every framework can be used for this method.

The final method is Quantization-Aware Training. The main feature of this method is that it is done during the retraining process. During it, the quantization effect will be simulated and new weights will be ready to undergo quantization. As simple static quantization models, inference time and model size should be decreased, but the QAT should minimize the loss in performance as it moves weights closer to the future clipping ranges, so the quantization will be more effective. However, this is also a framework and system-demanding method, so the same problem as with static quantization can be encountered, so Again not every machine and every framework can be used for this method.

4.3.2 Experimental protocol

The objective of this part is to decrease the model size, and inference time and not increase DER or increase by a small margin. For dynamic quantization, no additional steps are required. In the case of static quantization, the calibration step is done on the training subset of AMI-SDM. For QAT, the training will be done with the same training subset. Also, mixed quantization will be performed, where some layers will be quantized with static quantization, some with dynamic, and then a calibration step will be performed with usage of AMI-SDM training set. The results of the quantization will be evaluated on the AMI-SDM test subset based on several metrics namely DER, inference time, and model size.

TABLE 4.2: Quantization Methods for Speaker Diarization Model for Different Methods and Processing Units in Pytorch

Method	**Sincnet (Convolutional)**	**LSTM**	**Linear**	**CPU**	**GPU**
Dynamic	No	Yes	Yes	Yes	No
Static	Yes	No	Yes	Yes	Yes
Static with QAT	Yes	No	Yes	Yes	Yes

However, the important disadvantage is that not every quantization type can be performed on each type of layer. In Table 4.2 we can see that not every layer is possible to quantize for every type of quantization and processing unit which is a limitation of the PyTorch framework because of how the pre-trained model is distributed via the pyannote library, based on PyTorch.

For dynamic quantization, the available framework (PyTorch) allows only the dynamic quantization of two types of layers present in the speaker diarization model: Linear and LSTM. Therefore, all possible combinations will be tested, namely, only Linear layer quantized, only LSTM layer quantized, Linear + LSTM quantized, and then compared with the original model. For static quantization only Convolutional and Linear layers can be quantized, therefore only these two layers will be used during static quantization. However, dynamic quantization is possible for the LSTM layer, thus we will experiment with different combinations of quantized layers, perform calibration steps on a training set, and evaluate the performance.

Quantization-Aware Training has the same requirements as static quantization, so only Convolutional and Linear layers will be quantized as well as LSTM layers will be quantized dynamically.

4.3.3 Implementation details

Quantization combinations will be mixed and then acquired models will be trained for 1 epoch, using Adam Optimizer with a learning rate of 0.001 on the training subset. Then all results will be compared to the original one and evaluated by DER, inference time, and model size. All experiments were conducted on AWS EC2 instance which features NVIDIA T4 GPUs for high-performance computing, up to 25 Gbps of network bandwidth, and local NVMe SSD storage for fast data access.

4.3.4 Results

Our results are shown in Table A.1. The "Model with dynamically quantized linear layers" performs the best, achieving a DER of 27.5% on the test set" as well as the model with dynamically quantized LSTM layers. This model offers the lowest error rate among the listed options, which is what we aim to achieve. The static quantization methods increase the DER by a high margin showing that the variance of the weights during computation is too big. Therefore, the precomputation of statistics for the inference is not effective which resulted in a high DER of 64.7%. The QAT tried to solve the problems caused by static quantization with the calibration but the weight variance is still too large, so it was not effective enough, so DER is still too high: 49.5%.

In terms of inference time which was measured on 2s audio chunk, only the "Model with dynamically quantized linear layers" showed improvement. All other models have experienced increased inference time due to additional computations of the clipping range for LSTM layers as we think.

Considering the size, the best model is the mixed quantization model with a size of 1.5 MBs. In all tests, LSTM quantization shows a greater decrease in model size due to the fact that LSTM layers occupy the largest amount of space in this model.

TABLE 4.3: Quantization Impact on Model Size, Inference Time, and DER

Models	**Model Size (Mbs)**	**Inference Time (ms)**	**DER (all test files)**
Baseline (PyanNet)	5.9	11.6	**27.4%**
Dynamically quantized Linear	5.7	**9.7**	27.5%
Dynamically quantized LSTM	1.8	58.1	27.5%
Dynamically quantized Linear + LSTM	1.6	61.9	27.6%
Mixed quantization (Static conv, dynamic LSTM and Linear)	**1.5**	51.4	64.7%
Mixed QAT (Static conv + Linear, dynamic LSTM)	1.5	57.7	49.5%

In summary, the "Model with dynamically quantized linear layers" excels in achieving a lower DER and smaller inference time, while the quantization of both Linear and LSTM layers showcase smaller model sizes, but a worse inference time.

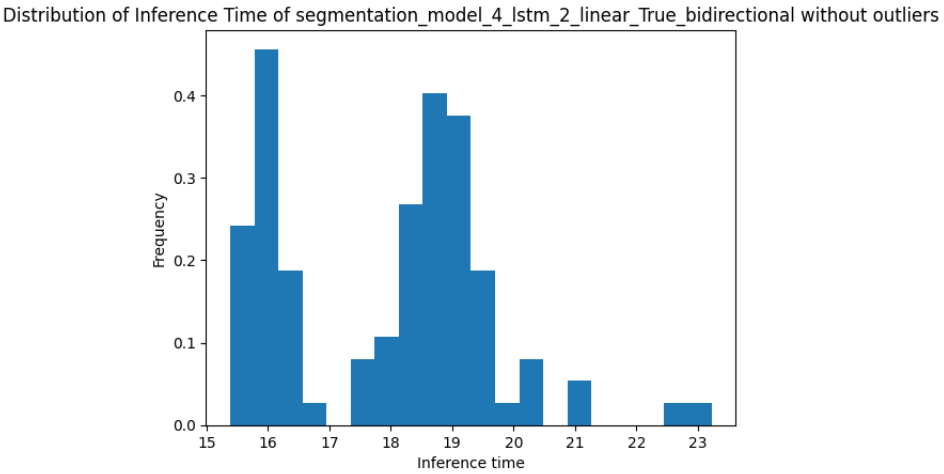

FIGURE 4.3: Inference without outliers

Depending on the specific requirements and priorities, one can choose the appropriate model based on the desired balance between DER, inference time, and model size. The increase in inference time for the LSTM layers quantization case is connected with the fact that the computation of clipping range each time increases inference time. In summary, the decrease of inference time by quantization is countered by the additional inference time of each calculation in LSTM layers. For the inference on edge devices model with a big decrease in size can be recommended, especially with dynamically quantized LSTM layers. However, the inference time of this model will be worse than the initial one.

4.4 Optimization by knowledge distillation

In this part of our experiments, we test the capabilities of knowledge distillation. The main goal is the same as before: obtain a smaller, faster model without loss in DER.

4.4.1 Methods

First, we start with changing the architecture of the initial model. Firstly, we will work with a number of LSTM layers in order to check whether 4 LSTM layers are too many for such a task and that fewer numbers will still have the same performance. Also, we want to test the hypothesis that making the LSTM monodirectional will reduce inference time and we want to know how much performance must be sacrificed to that end.

Another parameter that we want to test is the number of Linear layers. The main purpose is the exploration of model size reduction possibilities to understand their impact on performance. This should allow us to determine what kind of model we would distill for this task.

Finally, we will replace the Sincnet with the MFCC [32] extraction layer and try to test its performance due to the fact that MFCC can be cached in the online setting, contrary to SincNet features that can't be cached due to chunk-wise feature normalization, so this will decrease inference time and decrease the need in computation resources.

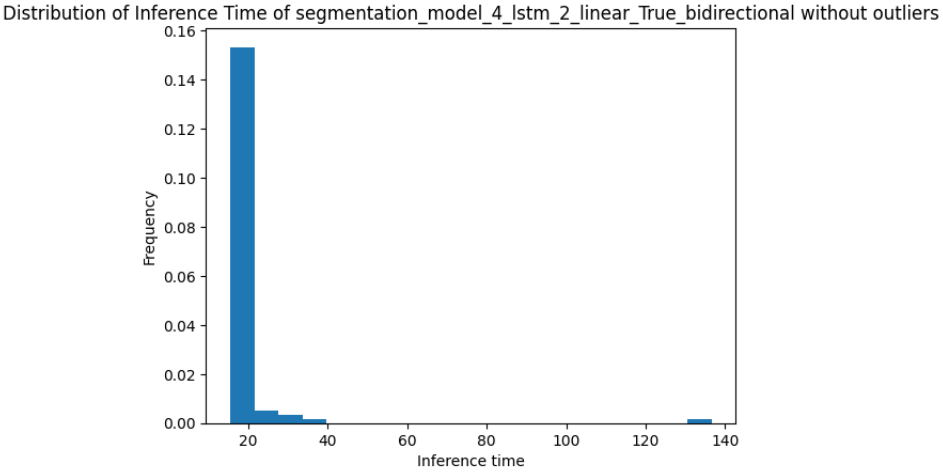

FIGURE 4.4: Inference with outliers

4.4.2 Experimental protocol

As we previously stated the main objective is to create a model better than the original in terms of inference time and model size, but without harming performance". For each iteration of experiments, we modify the model according to the iteration's architecture. For LSTM layers we will test each number of layers from 4 to 1, for the Linear layer the same operation from 2 to 1. In the case of MFCC, the number of MFCCs will be tested like 80, 120, 150. Then after the setup model will be trained and the best model of the training will be evaluated according to its chunk DER performance (average DER of all processed chunks) on the AMI-SDM development set. Then the inference time will be evaluated on the CPU based on 100 inferences on random audio chunks of length of 2 seconds as the average of these 100 inferences. Finally, the size of the model will be accessed and compared with the original one.

4.4.3 Implementation details

The training utilizes the baseline model as a teacher and the inputted probabilities to the student model won't be changed because the teacher outputs probabilities, so in cases where obviously two speakers present at the same segments will be transformed to the one only one speaker. The training will be conducted in 150 epochs with an Adam optimizer with a learning rate of 0.0001. The learning rate scheduler will be used which monitors DER on the development set of AMI-SDM with a patience of 5 and a factor of 0.5. Also, the early-stopping element with a patience of 15 was incorporated to prevent overfitting.

4.4.4 Results

As we can see from Figure 4.5 the number of LSTM layers with a combination of Linear layers doesn't influence the performance. Therefore, we can see that the inference time does decrease with the number of LSTM layers as well as with the number of Linear layers but the influence is not as strong as for LSTM. However, it can be clearly seen that the inference time dropped significantly when the type of LSTM layers changed from bidirectional to monodirectional. However, it had an influence on the performance which resulted in an increase in DER by 2 points which

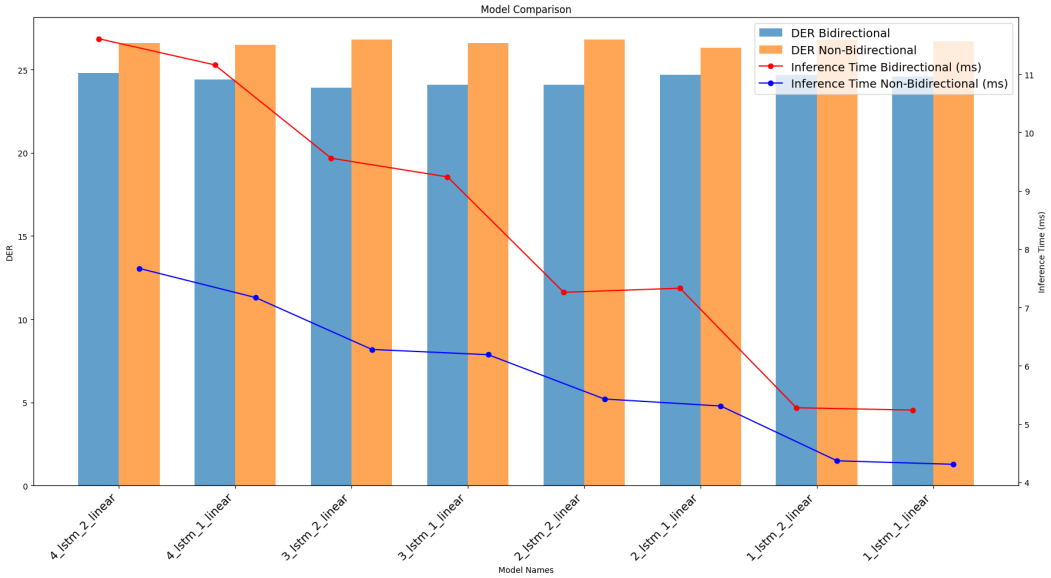

FIGURE 4.5: Knowledge distillation results on test set with sincnet with a comparison of different architectures

is noticeable. An interesting detail about inference time is that several first inferences take much longer time than others like in Figures 4.4 and 4.3. The full result with all details can be found in Appendix A.

The best model would be either bidirectional with 1 LSTM and 1 Linear layers model with DER of 26.7% on the development set with 4.31 ms inference time and model size decrease from 5.9 Mbs to 0.63 Mbs or monodirectional with the same number of layers with DER of 24.6% on the development set with 5.34 ms inference time and model size of 1.09 Mbs.

Next are the results of the training with the MFCC layer instead of sincnet. In Figure 4.6 it is visible that the best model in terms of a trade-off between DER and inference time is number 15 which is the model with 1 LSTM layer 1 Linear layer, bidirectional with 120 number of MFCC. The DER of this model is 27.5%, the inference time is 2.37 and the model size is 1.31 Mbs.

From this experiment, we can clearly see that the number of LSTM layers is not such a significant factor in terms of DER but for the inference time. The same can be said about Linear layers. However, the difference in terms of DER between bidirectional and monodirectional is visible. To summarize, the optimized model without a huge loss in DER are 1 LSTM layers model which has a solid decrease in inference time and model size.

In the case of MFCC, the speed-up of the model is significant, but the effectiveness of feature extraction by this algorithm is doubtful in cases of a small amount of MFCC. The number of MFCCs does play a big role in the general effectiveness of the model. However, whether the LSTM layers are bidirectional or monodirectional was also important in this setup. Thus, we can make a conclusion that the directionality of the LSTM layer is a determining factor for this model.

To summarize, we can confidently say that MFCC is a good replacement for sincnet if the number of MFCCs is not too low. Also, the performance in terms of DER highly relies on the type of LSTM layers rather than its amount. Finally, the amount of Linear layers doesn't play a big role in the performance of the model in

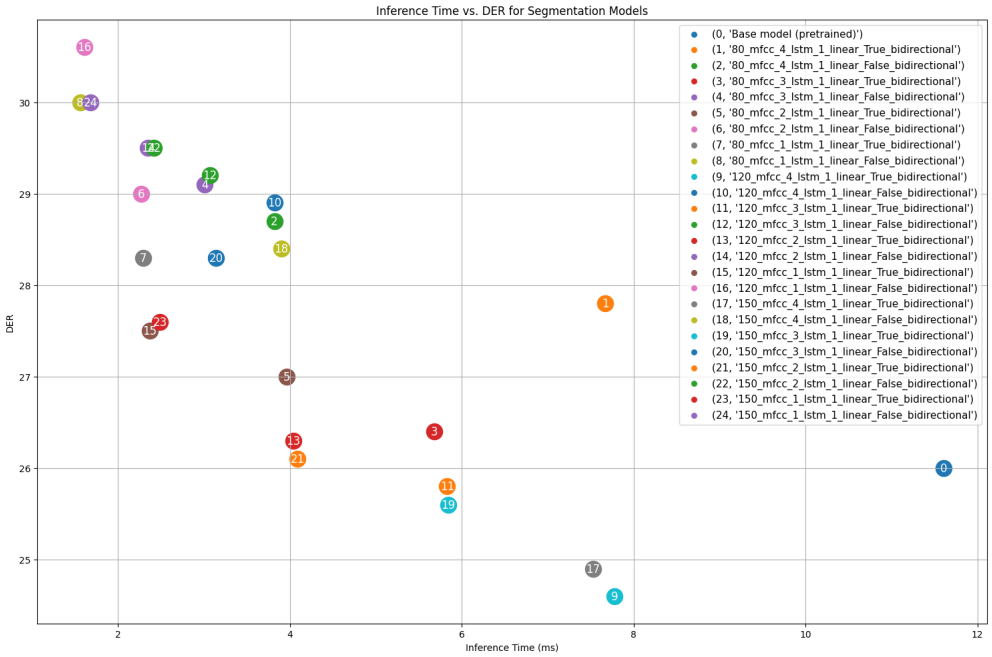

FIGURE 4.6: Knowledge distillation MFCC results in comparison on the development set DER vs Inference Time

terms of DER. For model usage on edge devices, the model should not be resource-demanding, slow, and have bad performance. Therefore, we determined 3 best models in this experiment, namely the bidirectional model with 1 LSTM and 1 Linear layers model with DER of 26.7% on the development set with 4.31 ms inference time and model size decrease from 5.9 Mbs to 0.63 Mbs or monodirectional with the same number of layers with DER of 24.6% on the development set with 5.34 ms inference time and model size of 1.09 Mbs with sincnet. In terms of MFCC the best model was the model with 1 LSTM layer 1 Linear layer, bidirectional with 120 MFCC, DER of 27.5%, the inference time of 2.37, and the model size is 1.31 Mbs.

4.5 Putting it all together

In the final part of the experiment, we will try to combine quantization and knowledge distillation methods in order to acquire the most optimized model which is ready for actual deployment of edge devices.

4.5.1 Methods

In this section, the only quantization method utilized will be dynamic. From previous experiments, we have tried several quantization methods but only this one yielded reasonable results without loss of DER and increase in inference time. Also, we want to combine methods that have already increased DER therefore, there is no need to increase it more.

From the knowledge distillation part, we will take all models that we acquired from previous testing. We want to obtain not only the most optimized model in terms of DER vs. Inference time and size trade-off but also the fastest model, smallest model for different application situations.

4.5.2 Experimental protocol

For the final selection, the 40 knowledge-distilled models are taken. Then all of them will be quantized dynamically in several ways: dynamic quantization of Linear or LSTM layers. All retrieved models will be evaluated based on three metrics: DER, inference time, and model size.

4.5.3 Implementation details

In terms of preprocessing no models need additional transformation before quantization. The best version of each layer configuration will be taken for this experiment. Subsequently, all of them will be quantized in two ways: only Linear layers and LSTM + Linear layers. This is done to evaluate how small can the model become and how efficient is quantization of Linear layers is.

All models will be compared to the initial one in terms of DER, latency, and model size. The DER is calculated on the development set of the AMI-SDM dataset, the inference time will be estimated based on 100 runs of 2-second chunks, and the model size will be measured as the size of the saved model.

4.5.4 Results

TABLE 4.4: Best models

Layers	Model size before quantization (Mbs)	Model size after only Linear quantization(Mbs)	Inference time before quantization(ms)	Inference time after Linear quantization	Model size after only Linear LSTM quantization(Mbs)	Inference time after Linear LSTM quantization(ms)	DER before quantization	DER after Linear quantization	DER after Linear+LSTM quantization
3_lstm_2_linear_True_bi	4.32	4.17	8.93	8.71	1.24	42.07	23.9%	23.9%	23.9%
3_lstm_2_linear_False_bi	1.75	1.66	5.46	5.45	0.58	22.15	26.8%	26.8%	26.68%
3_lstm_1_linear_True_bi	4.25	4.15	8.85	8.76	1.22	41.72	24.1%	24.1%	24.1%
2_lstm_1_linear_True_bi	2.67	2.57	6.50	6.43	0.82	28.99	24.1%	24.2%	24.2%
1_lstm_1_linear_True_bi	1.09	0.99	5.20	4.99	0.41	16.21	24.6%	24.7%	26.8%
80_mfcc_1_lstm_1_linear_True_bi	1.08	0.99	2.69	2.71	0.35	14.16	28.3%	28.5%	28.6%
120_mfcc_1_lstm_1_linear_True_bi	1.31	1.21	2.71	2.69	0.46	14.16	27.5%	27.5%	27.6%
150_mfcc_1_lstm_1_linear_True_bi	1.49	1.39	2.67	2.64	0.54	14.41	27.7%	27.6%	27.6%
1_lstm_1_linear_False_bi	0.63	0.58	4.29	4.16	0.29	9.96	26.7%	26.7%	26.7%
80_mfcc_1_lstm_1_linear_False_bi	0.59	0.54	1.65	1.70	0.22	7.67	30.0%	30.1%	30.2%
120_mfcc_1_lstm_1_linear_False_bi	0.73	0.68	1.92	1.86	0.30	7.86	30.6%	30.6%	30.6%
150_mfcc_1_lstm_1_linear_False_bi	0.85	0.80	2.00	1.85	0.38	7.80	30.0%	30.1%	30.1%

After obtaining all results we can see the best models on Table A.1. All results can be seen in the Appendix A. From the table, we can observe that some of the models even improved in terms of DER, all of them are faster after dynamic quantization with only Linear layers. They are much smaller after Linear+LSTM quantization, but inference time increases drastically. The best model in terms of a trade-off between DER and inference is with 120 MFCCs, 1 bidirectional LSTM layer, and 1 Linear layer. After all operations, its inference time is 2.69 ms, model size after linear quantization 1.21 Mbs, and DER 27.5%.

4.5.5 Discussion

To summarize, the combination of quantization and knowledge distillation produced useful results. We can see that there is flexibility in variants of optimization depending on one's goals. If the goal is to obtain the smallest model, then the model that one needs is the model with 60 MFCCs, 1 monodirectional LSTM, and 1 Linear

layer with the application of Linear + LSTM dynamic quantization. The obtained model is 0.22 Mbs which is almost 27 times smaller than the original model. For the fastest model, the right configuration is the same model but with just Linear quantization. If we want to have a balance between inference time, model size, and DER our choice would be a model with 120 MFCCs, 1 bidirectional LSTM layer, and 1 Linear with only Linear layer quantization. All of these models can be used in different situations depending on the goal. For example, if we want to utilize this model on different devices we can vary the models taking into account the device whether is powerful enough to execute the most accurate model or not, and the least demanding model should be used instead.

In conclusion, this work demonstrates the effectiveness of the combination of quantization and knowledge distillation for model optimization showing different configurations for different purposes. The model that was reduced by 27 times highlights the potential for resource saving. The obtained knowledge during this work is valuable for the one who is looking for a way to adapt models for different device configurations, promising effective deployment of the model in different scenarios.

Chapter 5

Conclusion

5.1 Summary

During this work, we studied two optimization methods: quantization, knowledge distillation, and a combination of them. We showed that knowledge distillation is a powerful tool for model optimization in speaker diarization which is as flexible as we can obtain models with qualities that we want. With this approach a model which is more capable than the teacher can be obtained. Also, the situation where the model is smaller, faster, and doesn't lose its performance is not uncommon. Therefore, this method can be called one of the best optimization tools in the optimization of speaker diarization and should definitely be taken into account to optimize models for the usage of edge devices.

The dynamic quantization on the other hand wasn't that effective for optimization purposes. As it was mainly designed to optimize standard matrix neural network computations, the computations of LSTM layers cause too many statistics computations which results in the increased inference time compared to an unquantized version of the same model. However, the quantization of Linear layers which is not the costliest part of the model has size effect smaller than that of inference time. In the case of static quantization, the speaker diarization model's weights have too high a variance to the precomputation of quantization parameters which results in high DER. The growth of DER from 26% to around 65% is too high, so this method cannot be recommended to optimize speaker diarization models. For QAT situation is better but not by a huge margin, the DER is around 50% is better than 65%, but still too much for this type of model. Problems with static quantization and QAT can be prevented with the changes in the model's architecture and with obtaining of bigger dataset for training and calibration.

Finally, we determined two working methods for optimization: dynamic quantization and knowledge distillation. Simultaneous utilization of these methods helps with obtaining optimized models of any kind: small, fast, or balanced in DER vs. Inference and Size trade-off. A model can be decreased in size by 30 times, the inference time can be reduced by 7 times, and DER can be dropped by around 2%. But everything comes at the cost of trade-offs between inference time and model size vs. DER, so at the end of the day we have to choose what we want: lower DER, lower inference time and model size, or decrease inference time and size until DER is acceptable.

5.2 Connection with MVA program

During my diploma research in speaker diarization optimization, the MVA program was certainly helpful and the knowledge and practice obtained during my education in this program was invaluable. Without information and programming skills obtained during the studies, this research would be challenging.

Firstly, one of the most important subjects was Computational Statistics. Not only did the mathematical knowledge help in understanding the mathematical algorithms that were utilized during this research but also helped with the interpretation of the obtained results during the experiments. Convex Optimization was really important as it is a core subject for understanding all optimization algorithms which was essential knowledge during the experiments. Also, Image Denoising knowledge was really useful during the research as I grasped lots of important mathematical concepts during these lectures like the Fourier Transform and Wavelet Transform can be used to separate noise from the signal in audio.

From the programming perspective, Deep Learning in Practice helped me dive deeper into different programming concepts of Neural Networks. I was able to use all obtained information in Deep Learning that was essential during the architecture process of the speaker diarization models, training them and interpreting the results. Also, Time Series Analysis was significantly important in my initial understanding of how the speaker diarization algorithms work as during this subject I obtained essential information for working with such type of time series as audio stream.

In summary, I was able to complete this internship of such complexity due to various concepts like integration of machine learning, mathematics, statistics, deep learning, and time series that were obtained during the scope of the year on the MVA program of ENS Paris-Saclay.

5.3 Future work

Despite reaching good results in the optimization of speaker diarization models there are several things that can be done during this research that should be studied in case of optimization. In the case of quantization, the bigger dataset with a wider range of training data will result in better quality of collected statistics as we have seen wasn't good enough. Therefore, resulting in better quantization results.

For knowledge distillation, one opportunity is to change the LSTM layer to a less complex one like RNN [45] or GRU [7] and experiment with whether the complexity of LSTM is necessary for the success of the model. If the result is satisfying the new model with RNN or GRU will be smaller and faster compared to the teacher model with LSTM layers. Also, the change of the LSTM layer will exclude the problem of quantization of the LSTM layer. This can lead to more valuable results in the experiments of simultaneous usage of two methods.

Finally, several methods could be used additionally to optimize the speaker diarization models. One of them is weight clustering which is assigning the weights with the same values or close to being stored as the same memory entity. This approach can lead to a decrease in model size, but no actual effect on inference time. The other one is pruning, or just removing the weights and connections that are close to zero to some extent (which we have to determine). This method will result in a smaller model size and faster inference as useless operations won't be executed.

Bibliography

[1] Waqar Ahmed et al. "Compact CNN Structure Learning by Knowledge Distillation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021, pp. 2–4.

[2] *AMI Corpus Overview*. https://groups.inf.ed.ac.uk/ami/corpus/overview.shtml. Accessed: September 12, 2023.

[3] Tom B. Brown et al. "A Review of Speaker Diarization: Recent Advances with Deep Learning". In: *Arxiv* 1.22 (July 2020), pp. 1–5. URL: https://arxiv.org/pdf/2005.14165.pdf.

[4] J. Carletta and et al. "The AMI meeting corpus: A pre-announcement". In: *International workshop on machine learning for multimodal interaction*. 2005, pp. 28–39.

[5] Z. Chen et al. "Block-Wise Embedding Adjustment for End-to-End Neural Diarization". In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 7419–7423.

[6] Z. Chen et al. "Online Recurrent Speaker-Aware Neural Network for Real-Time Diarization". In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 6669–6673.

[7] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *Université de Montréal* (2014). CIFAR Senior Fellow, p. 7.

[8] Juan M. Coria et al. "Overlap-Aware Low-Latency Online Speaker Diarization Based on End-to-End Local Segmentation". In: *ArXiv* 1 (Sept. 2021). URL: https://arxiv.org/pdf/2109.06483.pdf.

[9] M. Diez and L. Burget. "Analysis of DNN Embeddings for Speaker Diarization". In: *Proc. Interspeech*. 2019, pp. 3498–3502.

[10] Mireia Diez et al. "BUT System for DIHARD Speech Diarization Challenge 2018". In: *Proc. Interspeech 2018*. 2018, pp. 2798–2802.

[11] John Doe. "Calculating Clipping Range for Quantization". In: *Proceedings of the International Conference on Quantization Techniques*. 2021.

[12] G. Fiscus et al. "The Rich Transcription 2006 Spring Meeting Recognition Evaluation". In: *International Workshop on Machine Learning for Multimodal Interaction*. Springer, 2006, pp. 309–322.

[13] Yusuke Fujita et al. "End-to-End Neural Diarization: Reformulating Speaker Diarization as Simple Multi-label Classification". In: *Arxiv* 1.24 (Feb. 2020), pp. 2–3. URL: https://arxiv.org/pdf/2003.02966v1.pdf.

[14] A. Gholami et al. *A Survey of Quantization Methods for Efficient Neural Network Inference*. Tech. rep. University of California, Berkeley, 2018, pp. 2–8.

[15] Jianping Gou et al. "Knowledge Distillation: A Survey". In: (2023), pp. 5–6.

[16] D. Gsell and D. Garreau. "Joint Optimization of Permutation-Invariant Training and End-to-End Diarization". In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, pp. 7037–7041.

[17] Yuxian Gu et al. "Knowledge Distillation of Large Language Models". In: (2023), pp. 3–5.

[18] Maurice Günder, Nico Piatkowski, and Christian Bauckhage. "Full Kullback-Leibler-Divergence Loss for Hyperparameter-free Label Distribution Learning". In: *arXiv preprint arXiv:2209.02055* (2022), pp. 2–3.

[19] John Hershey et al. "Deep Clustering and Conventional Networks for Music Separation: Strong Together". In: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 2016.

[20] G. Hinton, O. Vinyals, and J. Dean. *Distilling the Knowledge in a Neural Network*. Tech. rep. Google Inc., Mountain View, 2015, p. 2.

[21] Shota Horiguchi et al. "Online Neural Diarization of Unlimited Numbers of Speakers Using Global and Local Attractors". In: *Arxiv* 1.22 (Dec. 2022), pp. 2–3. URL: https://arxiv.org/pdf/2206.02432.pdf.

[22] Shota Horiguchi et al. "Online Neural Diarization of Unlimited Numbers of Speakers Using Global and Local Attractors". In: *Arxiv* 1.22 (Dec. 2022), pp. 3–4. URL: https://arxiv.org/pdf/2206.02432.pdf.

[23] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *arXiv preprint arXiv:1712.05877* (2017), pp. 7–8.

[24] Benoit Jacob et al. "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference". In: *Arxiv* 1.15 (Dec. 2017), pp. 7–8. URL: https://arxiv.org/abs/1712.05877.

[25] *Keras Applications*. https://keras.io/api/applications/. Accessed: 2023-08-2.

[26] Federico Landini et al. "But System for the Second DIHARD Speech Diarization Challenge". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 6529–6533.

[27] Hyoje Lee et al. "Self-Knowledge Distillation via Dropout". In: 2022, pp. 3–4.

[28] Zechun Liu et al. "LLM-QAT: Data-Free Quantization Aware Training for Large Language Models". In: *Unpublished Manuscript* (2023), pp. 3–4.

[29] Y. Luo et al. "Online Speaker Diarization with Speaker-Tracing Buffer". In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 7414–7418.

[30] Y. Luo and N. Mesgarani. "TasNet: Surpassing Ideal Time-Frequency Magnitude Masking for Speech Separation". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27.8 (2018), pp. 1256–1266.

[31] Roy Miles and Krystian Mikolajczyk. "A closer look at the training dynamics of knowledge distillation". In: *arXiv preprint arXiv:2303.11098v3* (2023), pp. 5–7.

[32] L. Muda, M. Begam, and I. Elamvazuthi. "Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques". In: *Journal Name* (2010), p. 2.

[33] Keiron O'Shea and Ryan Nash. "A Review of Speaker Diarization: Recent Advances with Deep Learning". In: *Arxiv* 1.2 (Dec. 2015), pp. 3–8. URL: https://arxiv.org/pdf/1511.08458.pdf.

[34] D. S. Park et al. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Interspeech*. 2019, pp. 2613–2617.

[35] Tae Jin Parka et al. "A Review of Speaker Diarization: Recent Advances with Deep Learning". In: *Arxiv* 1.26 (Nov. 2021), pp. 1–2. URL: https://arxiv.org/pdf/2101.09624.pdf.

[36] PyTorch. *PyTorch Quantization Documentation*. Year Accessed. URL: https://pytorch.org/docs/stable/quantization.html.

[37] Pytroch. *Dynamic Quantizaiton*. https://pytorch.org/tutorials/recipes/recipes/dynamic_quantization.html. Accessed on September 13, 2023. 2023.

[38] Mirco Ravanelli and Yoshua Bengio. "Speaker Recognition from Raw Waveform with SincNet". In: *Mila, Université de Montréal* (2019). CIFAR Fellow, p. 2.

[39] M. T. T. Rodriguez et al. "Variational Bayesian Inference for Dirichlet Process Mixtures". In: *Proceedings of the 25th International Conference on Machine Learning*. ACM. 2008, pp. 872–879.

[40] Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. "A Comprehensive Survey on Model Quantization for Deep Neural Networks in Image Classification". In: (2020), pp. 7–8.

[41] Adriana Romero et al. "FITNETS: HINTS FOR THIN DEEP NETS". In: *arXiv* 1 (2015), p. 8. URL: https://arxiv.org/pdf/1412.6550.pdf.

[42] F. J. R. Ruiz et al. "Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2020, pp. 2075–2084.

[43] F. J. R. Ruiz et al. "Unsupervised Isolation Forest for Anomaly Detection on Time Series". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2019, pp. 2110–2118.

[44] G. Sell, A. Fischer, and F. Gouyon. "HMM-Based Multiple Speaker Diarization for Audio Segmentation and Indexing". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22.1 (2014), pp. 190–200.

[45] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network". In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020). Special Issue on Machine Learning and Dynamical Systems, p. 7.

[46] Kedeng Tong et al. "VRF: A Quantization-Error-Aware Variable Rate Framework for Learned Image Compression". In: (2019), p. 2.

[47] J. Wang et al. "Constraint Incremental Clustering: A Framework for Semi-Supervised Clustering". In: *IEEE Transactions on Knowledge and Data Engineering* 24.5 (2012), pp. 888–901.

[48] Xiaokai Wei et al. "Greener yet Powerful: Taming Large Code Generation Models with Quantization". In: *Proceedings of the ACM SIGPLAN International Conference on Software Engineering*. ACM. 2023, pp. 7–8.

[49] Han-Jia Ye, Su Lu, and De-Chuan Zhan. "Generalized Knowledge Distillation via Relationship Matching". In: 2022, pp. 3–6.

[50] Konrad Zuchniak. “Multi-teacher knowledge distillation as an effective method for compressing ensembles of neural networks”. In: (2022), pp. 33–35.

Appendix A

Quantization and knowledge distillation results

TABLE A.1: Quantization Impact before and after on Model Size, Inference Time, and DER

Layers	Model size before quantization (Mbs)	Model size after only Linear quantization(Mbs)	Inference time before quantization(ms)	Inference time after Linear quantization	Model size after only Linear LSTM quantization(Mbs)	Inference time after Linear LSTM quantization(ms)	DER before quantization	DER after Linear quantization	DER after Linear+LSTM quantization
4_lstm_2_linear_True_bidirectional	5.90	5.75	10.50	10.20	1.64	53.12	24.8%	24.9%	24.9%
4_lstm_2_linear_False_bidirectional	2.28	2.19	6.20	6.10	0.72	28.08	26.6%	26.6%	26.7%
4_lstm_1_linear_True_bidirectional	5.83	5.74	10.37	10.25	1.62	53.89	24.4%	24.5%	24.6%
80_mfcc_4_lstm_1_linear_True_bidirectional	5.83	5.73	7.85	7.61	1.56	51.93	27.8%	27.8%	27.8%
120_mfcc_4_lstm_1_linear_True_bidirectional	6.06	5.96	8.41	8.33	1.66	53.18	24.6%	24.7%	24.7%
150_mfcc_4_lstm_1_linear_True_bidirectional	6.24	6.14	7.94	7.83	1.75	53.04	24.9%	24.9%	24.9%
4_lstm_1_linear_False_bidirectional	2.22	2.17	6.52	6.62	0.70	28.65	26.5%	26.5%	26.6%
80_mfcc_4_lstm_1_linear_False_bidirectional	2.17	2.13	4.10	4.17	0.63	26.98	28.7%	28.8%	28.9%
120_mfcc_4_lstm_1_linear_False_bidirectional	2.32	2.27	3.99	3.90	0.71	27.31	28.9%	29.0%	29.0%
150_mfcc_4_lstm_1_linear_False_bidirectional	2.44	2.39	4.44	4.43	0.79	27.20	28.4%	28.5%	28.6%
3_lstm_2_linear_True_bidirectional	4.32	4.17	8.93	8.71	1.24	42.07	23.9%	23.9%	23.9%
3_lstm_2_linear_False_bidirectional	1.75	1.66	5.46	5.45	0.58	22.15	26.8%	26.8%	26.68%
3_lstm_1_linear_True_bidirectional	4.25	4.15	8.85	8.76	1.22	41.72	24.1%	24.1%	24.1%
80_mfcc_3_lstm_1_linear_True_bidirectional	4.25	4.15	6.32	6.10	1.16	39.85	26.4%	26.5%	26.5%
120_mfcc_3_lstm_1_linear_True_bidirectional	4.48	4.38	6.13	6.03	1.26	40.03	25.8%	25.9%	25.9%
150_mfcc_3_lstm_1_linear_True_bidirectional	4.65	4.56	6.83	7.30	1.35	40.17	25.6%	25.6%	25.6%
3_lstm_1_linear_False_bidirectional	1.69	1.64	5.94	5.87	0.57	22.69	26.6%	26.7%	26.8%
80_mfcc_3_lstm_1_linear_False_bidirectional	1.64	1.60	3.36	3.23	0.49	20.36	29.1%	29.2%	29.2%
120_mfcc_3_lstm_1_linear_False_bidirectional	1.79	1.74	3.56	3.48	0.58	20.43	29.2%	29.2%	29.2%
150_mfcc_3_lstm_1_linear_False_bidirectional	1.91	1.86	3.56	3.51	0.65	20.35	28.3%	28.3%	28.3%
2_lstm_2_linear_True_bidirectional	2.73	2.59	7.15	6.91	0.83	28.74	24.1%	24.2%	24.3%
2_lstm_2_linear_False_bidirectional	1.23	1.13	4.71	4.66	0.45	15.86	26.6%	26.6%	26.7%
2_lstm_1_linear_True_bidirectional	2.67	2.57	6.50	6.43	0.82	28.99	24.1%	24.2%	24.2%
80_mfcc_2_lstm_1_linear_True_bidirectional	2.67	2.57	4.60	4.47	0.76	27.07	27.0%	27.1%	27.1%
120_mfcc_2_lstm_1_linear_True_bidirectional	2.89	2.80	4.36	4.22	0.86	26.79	26.3%	26.5%	26.5%
150_mfcc_2_lstm_1_linear_True_bidirectional	3.07	2.98	4.84	4.72	0.94	27.19	26.1%	26.2%	26.2%
2_lstm_1_linear_False_bidirectional	1.16	1.11	4.73	4.63	0.43	16.25	26.8%	26.8%	26.9%
80_mfcc_2_lstm_1_linear_False_bidirectional	1.12	1.07	2.52	2.53	0.36	13.88	29.0%	29.1%	29.1%
120_mfcc_2_lstm_1_linear_False_bidirectional	1.26	1.21	2.51	2.47	0.45	14.15	29.5%	29.5%	29.6%
150_mfcc_2_lstm_1_linear_False_bidirectional	1.38	1.33	2.91	2.81	0.52	14.20	27.6%	27.6%	27.6%
1_lstm_2_linear_True_bidirectional	1.15	1.01	5.26	5.12	0.43	16.46	24.7%	24.7%	24.7%
1_lstm_2_linear_False_bidirectional	0.70	0.60	4.36	4.26	0.31	9.95	26.8%	26.8%	26.9%
1_lstm_1_linear_True_bidirectional	1.09	0.99	5.20	4.99	0.41	16.21	24.6%	24.7%	26.8%
80_mfcc_1_lstm_1_linear_True_bidirectional	1.08	0.99	2.69	2.71	0.35	14.16	28.3%	28.5%	28.6%
120_mfcc_1_lstm_1_linear_True_bidirectional	1.31	1.21	2.71	2.69	0.46	14.16	27.5%	27.5%	27.6%
150_mfcc_1_lstm_1_linear_True_bidirectional	1.49	1.39	2.67	2.64	0.54	14.41	27.7%	27.6%	27.6%
1_lstm_1_linear_False_bidirectional	0.63	0.58	4.29	4.16	0.29	9.96	26.7%	26.7%	26.7%
80_mfcc_1_lstm_1_linear_False_bidirectional	0.59	0.54	1.65	1.70	0.22	7.67	30.0%	30.1%	30.2%
120_mfcc_1_lstm_1_linear_False_bidirectional	0.73	0.68	1.92	1.86	0.30	7.86	30.6%	30.6%	30.6%
150_mfcc_1_lstm_1_linear_False_bidirectional	0.85	0.80	2.00	1.85	0.38	7.80	30.0%	30.1%	30.1%

TABLE A.2: Knowledge Distillation with SincNet

Model	Dev (%)	Test (%)	Size (Mbs)	GPU (min)	CPU Outliers (ms)	STD Outliers (ms)	CPU no Outliers (ms)	STD no Outliers (ms)
Base model	26.0	27.4	5.90	1.95	12.71	8.21	11.61	0.71
4_lstm_2_T	24.8	26.3	5.90	1.95	12.71	8.21	11.61	0.71
4_lstm_2_F	26.6	27.8	2.29	1.62	7.78	1.09	7.67	1.01
4_lstm_1_T	24.4	26.1	5.84	1.88	11.24	0.40	11.16	0.13
4_lstm_1_F	26.5	27.7	2.22	1.62	7.34	1.07	7.17	0.58
3_lstm_2_T	23.9	25.3	4.32	1.76	9.68	0.60	9.56	0.13
3_lstm_2_F	26.8	28.5	1.76	1.58	6.35	0.40	6.28	0.14
3_lstm_1_T	24.1	25.7	4.25	1.75	9.33	0.52	9.24	0.13
3_lstm_1_F	26.6	27.9	1.70	1.57	6.30	0.61	6.19	0.41
2_lstm_2_T	24.1	25.8	2.74	1.64	7.30	0.25	7.26	0.13
2_lstm_2_F	26.8	28.4	1.23	1.53	5.46	0.20	5.43	0.11
2_lstm_1_T	24.7	26.2	2.67	1.63	7.46	0.75	7.33	0.50
2_lstm_1_F	26.3	27.5	1.16	1.52	5.38	0.56	5.31	0.42
1_lstm_2_T	24.7	26.0	1.16	1.52	5.31	0.17	5.28	0.09
1_lstm_2_F	26.8	28.0	0.70	1.48	4.45	0.36	4.37	0.12
1_lstm_1_T	24.6	26.0	1.09	1.52	5.34	0.54	5.24	0.29
1_lstm_1_F	26.7	28.0	0.63	1.47	4.33	0.11	4.31	0.07

TABLE A.3: MFCC Knowledge Distillation

Model	Dev (%)	Best Test (%)	Size (Mbs)	GPU (min)	CPU Outliers (ms)	STD Outliers (ms)	CPU no Outliers (ms)	STD no Outliers (ms)
Base model (pretrained)	26.0	27.4	5.90	1.95	12.71	8.21	11.61	0.71
segmentation_model_80_mfcc_4_lstm_1_linear_True_bidirectional	27.8	29.2	5.83	4.69	1.58	8.20	7.67	0.23
segmentation_model_80_mfcc_4_lstm_1_linear_False_bidirectional	28.7	29.9	2.18	0.34	1.33	3.87	3.82	0.26
segmentation_model_80_mfcc_3_lstm_1_linear_True_bidirectional	26.4	27.2	4.25	0.30	1.45	5.73	5.68	0.09
segmentation_model_80_mfcc_3_lstm_1_linear_False_bidirectional	29.1	30.5	1.64	0.17	1.29	3.04	3.01	0.04
segmentation_model_80_mfcc_2_lstm_1_linear_True_bidirectional	27.0	27.5	2.67	0.38	1.35	4.02	3.96	0.27
segmentation_model_80_mfcc_2_lstm_1_linear_False_bidirectional	29.0	30.1	1.12	0.14	1.24	2.30	2.27	0.05
segmentation_model_80_mfcc_1_lstm_1_linear_True_bidirectional	28.3	29.8	1.08	0.12	1.24	2.31	2.29	0.04
segmentation_model_80_mfcc_1_lstm_1_linear_False_bidirectional	30.0	29.9	0.59	0.12	1.20	1.58	1.56	0.07
segmentation_model_120_mfcc_4_lstm_1_linear_True_bidirectional	24.6	26.2	6.06	0.44	1.59	7.85	7.78	0.23
segmentation_model_120_mfcc_4_lstm_1_linear_False_bidirectional	28.9	30.0	2.32	0.21	1.33	3.86	3.82	0.06
segmentation_model_120_mfcc_3_lstm_1_linear_True_bidirectional	25.8	26.2	4.48	0.49	1.46	5.94	5.83	0.12
segmentation_model_120_mfcc_3_lstm_1_linear_False_bidirectional	29.2	29.8	1.79	0.21	1.29	3.11	3.07	0.04
segmentation_model_120_mfcc_2_lstm_1_linear_True_bidirectional	26.3	26.2	2.89	0.14	1.36	4.06	4.04	0.06
segmentation_model_120_mfcc_2_lstm_1_linear_False_bidirectional	29.5	30.1	1.26	0.15	1.24	2.38	2.35	0.03
segmentation_model_120_mfcc_1_lstm_1_linear_True_bidirectional	27.5	27.4	1.31	0.08	1.24	2.39	2.37	0.03
segmentation_model_120_mfcc_1_lstm_1_linear_False_bidirectional	30.6	31.4	0.73	0.04	1.20	1.61	1.61	0.02
segmentation_model_150_mfcc_4_lstm_1_linear_True_bidirectional	24.9	26.3	6.24	0.35	1.60	7.60	7.53	0.13
segmentation_model_150_mfcc_4_lstm_1_linear_False_bidirectional	28.4	29.5	2.44	0.18	1.34	3.94	3.90	0.07
segmentation_model_150_mfcc_3_lstm_1_linear_True_bidirectional	25.6	26.3	4.65	0.22	1.46	5.88	5.84	0.09
segmentation_model_150_mfcc_3_lstm_1_linear_False_bidirectional	28.3	29.4	1.91	0.12	1.30	3.17	3.14	0.04
segmentation_model_150_mfcc_2_lstm_1_linear_True_bidirectional	26.1	27.0	3.07	0.31	1.37	4.15	4.09	0.05
segmentation_model_150_mfcc_2_lstm_1_linear_False_bidirectional	29.5	29.2	1.38	0.17	1.25	2.46	2.42	0.07
segmentation_model_150_mfcc_1_lstm_1_linear_True_bidirectional	27.6	27.7	1.49	0.14	1.26	2.51	2.49	0.05
segmentation_model_150_mfcc_1_lstm_1_linear_False_bidirectional	30.0	30.1	0.85	0.08	1.21	1.70	1.68	0.04