University of Sarajevo
Faculty of Electrical Engineering
Department of Computer Science and Informatics

Analysis of Early Media in Voice Communication

Bachelor's thesis

- First cycle of studies -

Student:
Faruk Zahiragić

Mentor:
Doc. dr Senka Krivić

Sarajevo,
July 2025

Sažetak

Rana medija u SIP komunikaciji predstavlja mogućnost dva SIP agenta da komuniciraju prije nego što SIP poziv bude uspostavljen. Čest problem koji ponuđači glasovnih servisa imaju je da ne mogu zaključiti razlog neuspostavljana SIP poziva. Jedan od načina otkrivanja uzroka neuspostavljana SIP poziva je određivanje vrste rane medije poziva. Ovaj rad upravo pokušava riješiti navedeni problem tako što klasificira ranu mediju i na osnovu nje određuje izvor neuspostavljanja poziva. Uvođenjem ekstenzivnijih datasetova audio podataka kao što je *AudioSet* otvorila se mogućnost primjene neuronskih mreža koje zahtjevaju takve količine podataka. Iz toga razloga u ovome radu se istražuje primjena distilacije znanja iz jednog takvog modela na resursno nezahtjevan model prikladan za praktičnu upotrebu. Model vrši klasificiranje ranih medija u četiri klase: *music*, *silence*, *speech* i *ringing*. Ovakav model se pokazao pouzdanim za ovu primjenu ostvarujući visok stepen preciznosti. S tim u vidu ova analiza bi mogla postaviti temelj za dalji razvoj tehnika klasifikacije *audio* podataka, kao što je distilacija znanja iz *foundation* modela.

Ključne riječi: rana medija, distilacija znanja, klasifikacija, *AudioSet*

Abstract

Early media in SIP communication represents the possibility of two SIP agents to communicate before a SIP call is established. A common problem that voice service providers have is that they cannot determine the reason for the failure of a SIP call. One way to detect the cause of a failed SIP call is to determine the type of early media in the call. This paper attempts to solve this problem by classifying early media and, based on it, determining the source of the call failure. The introduction of more extensive audio data datasets, such as AudioSet, opens up the possibility of applying neural networks that require such amounts of data. For that reason, this paper explores the application of knowledge distillation from such a model to a resource-constrained model suitable for practical use. The model classifies early media into four classes: Music, silence, speech, and ringing. This model has proven to be reliable for this application and achieves a high degree of accuracy. With this in mind, this analysis could lay the foundation for further development of audio data classification techniques, such as knowledge distillation from foundation models.

Keywords: early media, knowledge distillation, classification, *AudioSet*

Elektrotehnički fakultet, Univerzitet u Sarajevu
Odsjek za računarstvo i informatiku
Doc. dr. Senka Krivić
Sarajevo, Juli 2025

Postavka zadatka završnog rada I ciklusa: **Analiza ranih medija u glasovnoj komunikaciji**

Ovaj završni rad ima za cilj da analizira mogućnosti primjene distilacije znanja u klasifikaciji ranih medija u glasovnoj komunikaciji. Engl. *Early Media* u SIP komunikaciji je mogućnost dva SIP agenta da komuniciraju prije nego što SIP poziv bude uspostavljen. Čest problem koji ponuđači glasovnih servisa imaju je da ne mogu zaključiti razlog neuspostavljana SIP poziva. Jedan od načina otkrivanja uzroka neuspostavljana SIP poziva je određivanje vrste rane medije poziva. Ovaj rad upravo pokušava riješiti navedeni problem tako što klasificira ranu mediju i na osnovu nje određuje izvor neuspostavljanja poziva.

Polazna literatura:

[1] Altwlkany, Kemal, et al. "Knowledge Distillation for Real-Time Classification of Early Media in Voice Communications." 2024 32nd International Conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, 2024.

[2] Kong, Qiuqiang, et al. "Panns: Large-scale pretrained audio neural networks for audio pattern recognition." IEEE/ACM Transactions on Audio, Speech, and Language Processing 28 (2020): 2880-2894.

[3] Gemmeke, Jort F., et al. "Audio set: An ontology and human-labeled dataset for audio events." 2017 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2017.

Doc. dr. Senka Krivić, dipl. ing. el.

Univerzitet u Sarajevu
Elektrotehnički fakultet
Odsjek za računarstvo i informatiku

Izjava o autentičnosti radova

Završni rad
I ciklusa studija

Ime i prezime: Faruk Zahiragić
Naslov rada: Analysis of Early Media in Voice Communication
Vrsta rada: Završni rad Prvog ciklusa studija
Broj stranica: 47

Potvrđujem:

- da sam pročitao dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- da sam svjestan univerzitetskih disciplinskih pravila koja se tiču plagijarizma;
- da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznačeno;
- da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- da sam jasno naznačio prisustvo citiranog ili parafraziranog materijala i da sam se referirao na sve izvore;
- da sam dosljedno naveo korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- da sam odgovarajuće naznačio svaku pomoć koju sam dobio pored pomoći mentora i akademskih tutora/ica.

Sarajevo, juli 2025

Potpis:

____

Faruk Zahiragić

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem description and motivation

Although the growth of mobile phone usage in developed countries has stagnated in recent years, emerging markets have only recently fully utilized this technology. However, the rapid growth of mobile phone usage in developing countries was not followed by legal legislation from mobile network operators (MNOs) in these markets [9]. This problem is exacerbated for Voice over IP (VoIP) technology, which is still growing in popularity in the aforementioned markets. VoIP calls are usually initiated using the Session Initiation Protocol (SIP). SIP, much like HTTP, uses standardized response codes as a way to inform the caller of the status of the call. However, given the fact that MNOs are the ones that determine the response code, these codes don't have to be a real representation of the call status, as MNOs can return false responses, considering they are often not legally obliged to do so in developing markets.

Voice service providers are companies that provide various types of programmable VoIP calls, which are used, among other usages, to perform individual or campaign-type calls to customers, usually for marketing purposes. Voice service providers heavily utilize SIP response codes when making phone calls to customers. A common use case is determining whether to perform a redial when a call fails to be established. For this purpose, they first determine the reason for the call failure via the SIP response code, and upon determining if the call ended with the callee being busy, performing the redial, or not redialing in case of a network error. This logic cannot be utilized if the telecom provider did not return a proper SIP response code, prompting a need for Voice Service providers to determine the status of the failed call in a different way.

1.2 Proposed solution

To mitigate the aforementioned issue, the proposed approach is to analyze the call before it is established in order to determine the cause of the call failure. Since the calls failed to be established, the only information that is sent from the callee to the caller is early media. Early media is defined as the audio that is transferred in a VoIP call before the call is established [10]. This audio is usually generated by the operator, and it usually differs depending on the status of the call.

Most early media are standardized by operators based on the status of the call. So, for example, when the call fails to be established due to network congestion, the early media is expected to be the same for that reason. By collecting this characteristic early media and mapping it with a SIP response code, the equivalent SIP response code for future failed calls could be determined by comparing it with the already collected early media. This comparison could be performed with acoustic fingerprinting techniques [1]. However, performing this comparison is time-intensive, which, for the time constraints of real-time communication, is a major issue. To mitigate this issue, the mean time required for acoustic fingerprinting could be reduced by detecting if the early media contains any speech and, only if that is the case, performing acoustic fingerprinting. This detection could be achieved by performing audio tagging of the call, with the constraint that the audio tagging be achieved within an optimal amount of time. To achieve this, the model used for performing audio tagging should not be resource-intensive, and perform the inference in a short time frame. To achieve fast inference while retaining an acceptable amount of accuracy, knowledge distillation from a proprietary audio tagging model will be used as the teacher, while an efficient student model based on the random forest algorithm will be used. The proposed solution will be developed in phases as seen in figure 1.1

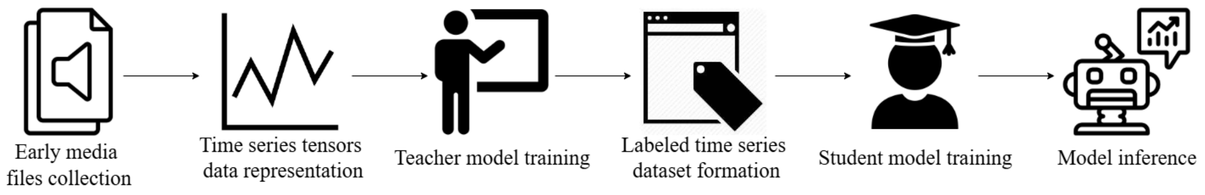


Figure 1.1: Knowledge distillation process

1.3 Methodology

In order to implement the required model, several steps have to be taken to ensure the completeness of the solution:

- **Literature Review:** Conducting a comprehensive review of existing audio tagging and knowledge distillation models, as well as work related to VoIP communication.
- **Model Selection:** The teacher model used is an implementation of a PANN (Pretrained Audio Neural Networks). This model is chosen because of its state-of-the-art performance when compared to other contemporary audio tagging models. The student model used is an implementation of a random forest classifier chosen for its fast inference.
- **Data Preparation:** Curating a dataset made up of early media audio files. The dataset needs significant preprocessing before feeding it to the student model, which will also be done in this step.
- **Model development and training:** Perform knowledge distillation using the chosen teacher and student models.
- **Model evaluation:** Evaluate the achieved results of the trained model.
- **Conclusion and Recommendations:** The thesis will conclude with a comprehensive summary of the work carried out and the results achieved. It will also provide insights into potential areas for future research and enhancements to the classification model.

1.4 Expected results

The expected results of this thesis are to successfully implement an Early Media Classification model. The model is expected to perform the classification significantly faster than using a resource-intensive neural network-based model, while retaining similar accuracy to state-of-the-art audio taggers.

1.5 Outline

The following is an outline of the thesis separated by chapters:

- **Chapter 1** introduces the concept of Early Media in Voice Communication.
- **Chapter 2** reviews the existing body of literature in the field of Voice Communication and Knowledge Distillation, highlighting the development of these fields for solving specific problems like Early Media Classification.
- **Chapter 3** delves into the core of research, investigating how Knowledge Distillation is utilized for audio tagging.
- **Chapter 4** is dedicated to the analysis of datasets used to train the teacher and the student models.
- **Chapter 5** is focused on the implementation of the model, from teacher model inference to training the student model.
- **Chapter 6** unveils the results achieved with the development of the model. A comparative analysis of accuracy and inference time for the student and teacher models is given.
- **Chapter 7** concludes the investigation, where key findings are synthesized and possible future research areas are emphasized.

Chapter 2

Literature review

2.1 Voice Communication

With the rise of the Internet, new ways of voice communication, such as VoIP, were introduced. During the early development of VoIP, various patents were filed and innovations discovered, all condensedly explained by B. Goode (2002) [11] in his work on this topic. Schulzrinne et al. standardized VoIP with SIP (2002) [12] as a protocol that accommodates the specific states of a voice communication session. Most elements of SIP are defined with a Request for Comment (RFC). An RFC describes the technical foundations of a given internet-related term [13]. A major term in SIP communication relevant to this thesis is Early Media, which was defined by Camarillo, G., and H. Schulzrinne in RFC 3960 (2004) [10]. They specified the technical details of Early Media, such as its position in a SIP call and the various models of implementation of Early Media.

2.2 Audio Classification

Early research in the field of audio classification used statistical approaches to perform the comparison. Bugatti et al. (2002) [14] compared using an approach based on Bayesian classification and zero-crossing rate with a multi-layer Perceptron for classifying audio into speech and music. As expected, the neural network approach was more resource-demanding and accurate, while the statistical approach proved to be more resource-efficient; however, its performance degraded when encountering more complex input, such as speech superimposed on music.

Further work on efficient audio classification models was made by Lavner Yizhar and Dima Ruinskiy (2009) [15]. They used a decision-tree-based algorithm to perform the classification. This proved to be highly accurate in simple cases; however, identification rates degraded for classifying transitioning audio from one class to another and superimposed audio from different classes.

Audio classification has seen remarkable advancements through deep learning methodologies, particularly in speech recognition, audio classification, and audio event detection. Performing audio classification via deep learning methods requires extensive datasets of diverse audio for training. For that purpose, datasets like the MagnaTagATune [16] and Million Song Dataset [17] were collected. Using those datasets, Keunwoo Choi et al. (2016) [18] trained deep convolutional neural networks for the purpose of audio tagging. This approach proved to be useful for tagging the audio with 50 possible labels.

Another more extensively researched approach to deep learning methods for audio classification analyzed the application of existing computer vision models and adapted them for audio classification. One such application was performed by Shawn Hershey et al. [19]. They analyzed the performance of various convolutional neural network architectures developed for image classification tasks on large-scale audio classification. Experiments were performed by evaluating the performance of these specialized CNNs against a baseline of a general fully connected model. Results show that these CNNs are effective in audio classification and that larger training and label sets improve the accuracy up to a certain extent.

Large-scale audio dataset collection in recent years has given way to the ability to develop purpose-built deep learning models for audio classification. The most significant of these datasets is AudioSet, a large-scale dataset of manually-annotated audio events that endeavors to bridge the gap in data availability between image and audio research [3].

Two of the most prominent models trained on AudioSet are Yet Another Mobile Network (YAMNet) [20] and Pretrained Audio Neural Networks (PANNs) [2]. Mohammed et al. introduced a transfer model from YAMNet that was successfully used to classify drones using radio frequencies [21]. This revelation suggests such models would make for quality teacher models for knowledge distillation tasks as well. One of the most prominent models is Pretrained Audio Neural Networks (PANNs). In their paper, Qiuqiang Kong et al. introduced PANNs as a collection of neural networks designed to enhance various audio pattern recognition tasks. Various novel architectures were analyzed, but the most relevant one for this observation that would be used in the implementation of the model is a 14-layered convolutional neural network named CNN14. CNN14 achieved a state-of-the-art mean average precision (mAP) of 0.439. PANNs were also shown to be capable of generalizing to other audio pattern recognition tasks [2].

2.3 Knowledge Distillation

Knowledge distillation was first introduced by Hinton et. al. (2015) [22] as a novel framework for training efficient machine learning models. This work led the way to the development of a plethora of teacher-student architectures and applications, which Gou et al. (2021) surveyed in detail [7].

2.4 Applications of Audio Classification to Voice Communication

A related problem to Early Media classification is Answering Machine Detection (AMD). AMD seeks to determine whether a phone call was answered by a machine or a human. Ansimov et al. implemented an AMD model using convolutional neural networks [23]. Altwlkany et al. [24] implemented AMD using transfer learning from YAMNet that proved to be effective for real-time constraints in voice communication.

Using existing neural networks specifically trained for audio classification on large datasets like AudioSet allowed for knowledge distillation on specific tasks like early media detection. A relevant application of one such model was introduced by Kemal Altwlkany et al. [1] by examining knowledge distillation from a PANN model onto a high-performing gradient-boosted tree (GBT) model trained on a dataset consisting of early media audio files.

2.5 Conclusion

This thesis primarily relies on the early media classification findings by Kemal Altwlkany et al., with the primary difference being the student model used in the knowledge distillation process. Namely, the student model chosen for this thesis is a random forest classifier, chosen because of its proven reliability for audio event detection by the works of Xia, Xianjun, et al. [25] and alike.

Chapter 3

The Problem of Early Media Classification

Before diving into the implementation details of the model that is being developed, it is necessary to explain the concepts behind the proposed solution.

3.1 Voice over IP

Voice over Internet Protocol (VoIP), also known as IP telephony, is a set of technologies used primarily for voice communication sessions over Internet Protocol (IP) networks, such as the Internet [26]. VoIP enables voice calls to be transmitted as data packets, facilitating various methods of voice communication, including traditional applications like Zoom, Microsoft Teams, Google Voice, and VoIP phones. Regular telephones can also be used for VoIP by connecting them to the Internet via analog telephone adapters (ATAs), which convert traditional telephone signals into digital data packets that can be transmitted over IP networks. Figure 3.1 represents a basic implementation of a VoIP network.

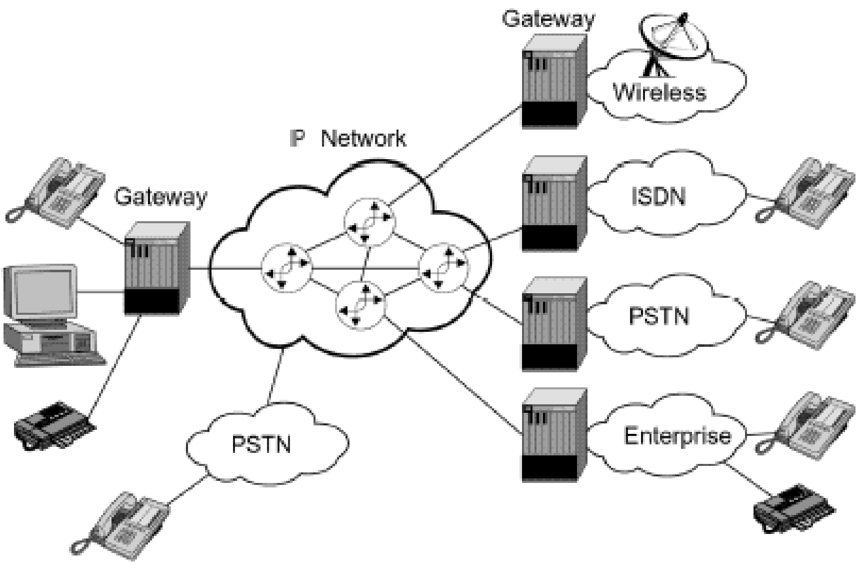


Figure 3.1: A basic implementation of VoIP [4]

3.1.1 Session Initiation Protocol

The Session Initiation Protocol (SIP) is a signaling protocol used for initiating, maintaining, and terminating communication sessions that include voice, video, and messaging applications [27]. SIP is used in Internet telephony, in private IP telephone systems, as well as mobile phone calling over long-term evolution (VoLTE).

The network elements that use the Session Initiation Protocol for communication are called SIP user agents. Each user agent (UA) performs the function of a user agent client (UAC) when it is requesting a service function, and that of a user agent server (UAS) when responding to a request.

SIP is a text-based protocol with syntax similar to that of HTTP. There are two different types of SIP messages: requests and responses. The first line of a request has a method, defining the nature of the request, and a Request Uniform Resource Identifier (URI), indicating where the request should be sent. The first line of a response has a response code.

SIP Requests initiate a functionality of the protocol. They are sent by a user agent client to the server and are answered with one or more SIP responses, which return a result code of the transaction, and generally indicate the success, failure, or other state of the transaction.

The most commonly used SIP requests are:

- **INVITE:** Initiate a dialog for establishing a call. The request is sent by a user agent client to a user agent server.
- **ACK:** Confirm that an entity has received a final response to an INVITE request.
- **BYE:** Signal termination of a dialog and end a call.

Responses are sent by the user agent server, indicating the result of a received request. Several classes of responses are recognized, determined by the numerical range of result codes:

- **1xx**: Provisional responses to requests indicate the request was valid and is being processed.
- **2xx:** Successful completion of the request. As a response to an INVITE, it indicates that a call is established. The most common code is 200, which is an unqualified success report.
- **3xx:** Call redirection is needed for completion of the request. The request must be completed with a new destination.
- **4xx:** The request cannot be completed at the server for a variety of reasons, including bad request syntax (code 400).
- **5xx:** The server failed to fulfill an apparently valid request, including server internal errors (code 500).
- **6xx:** The request cannot be fulfilled at any server. It indicates a global failure, including call rejection by the destination.

A common case in practice is to have multiple UA proxies between end users that are trying to establish a call between each other, as presented in figure 3.2.

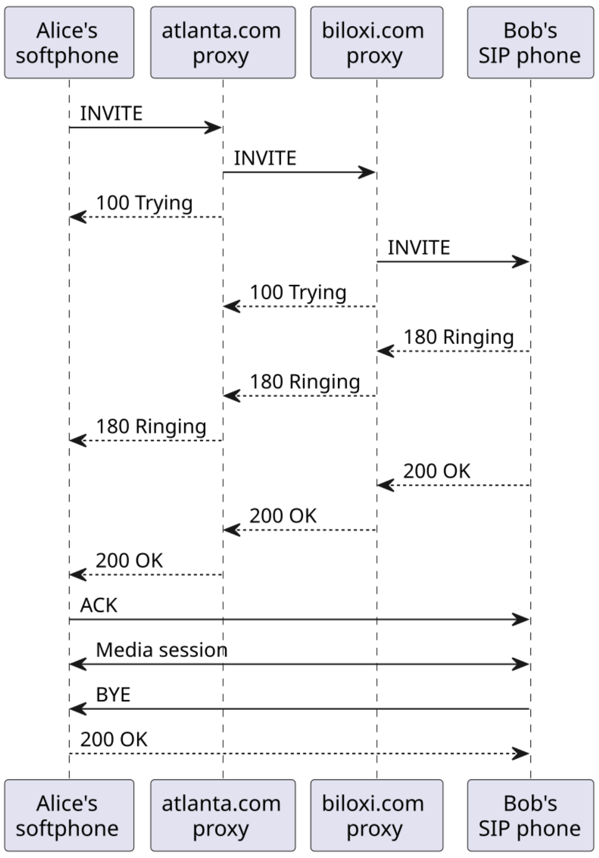


Figure 3.2: An example of a SIP session setup [5]

3.1.2 Early Media

Early media refers to media (e.g., audio and video) that is exchanged before a particular session is accepted by the called user [10]. Within a dialog, early media occurs from the moment the initial INVITE is sent until the User Agent Server generates a final response. It may be unidirectional or bidirectional, and can be generated by the caller, the callee, or both. Typical examples of early media generated by the callee are ringing tone and announcements (e.g., queuing status). Early media generated by the caller typically consists of voice commands or dual tone multi-frequency (DTMF) tones to drive interactive voice response (IVR) systems. The position of Early Media in SIP signalization is visible in figure 3.3.

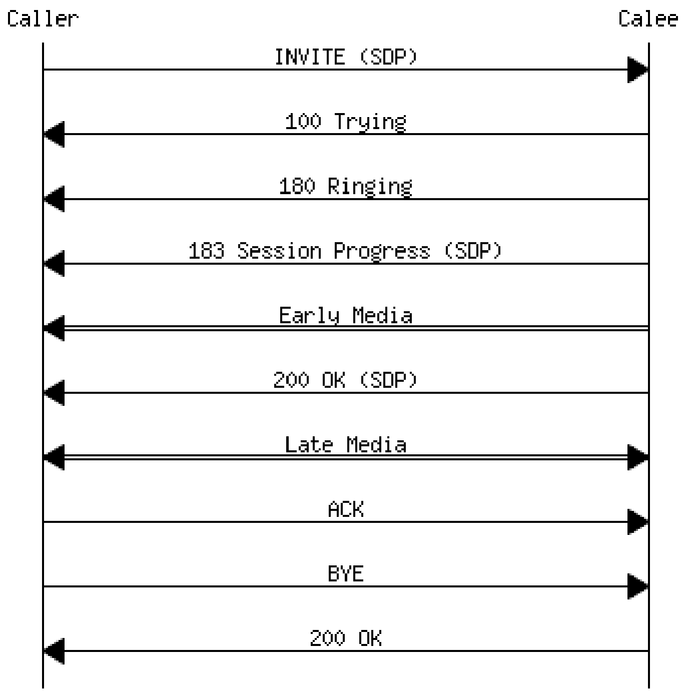


Figure 3.3: Early media in a SIP session setup [6]

3.2 Knowledge Distillation

Knowledge distillation in machine learning is the process of transferring knowledge from a large model to a smaller one [22]. While large models (such as very deep neural networks or ensembles of many models) have more knowledge capacity than small models, this capacity might not be fully utilized. It can be just as computationally expensive to evaluate a model even if it utilizes little of its knowledge capacity. Knowledge distillation transfers knowledge from a large model to a smaller one without loss of validity. As smaller models are less expensive to evaluate, they can be deployed on less powerful hardware and perform inference faster.

Knowledge transfer from a large model to a small one somehow needs to teach the latter without loss of validity. If both models are trained on the same data, the smaller model may have insufficient capacity to learn a concise knowledge representation compared to the larger model. However, some information about a concise knowledge representation is encoded in the pseudolikelihoods assigned to its output: when a model correctly predicts a class, it assigns a large value to the output variable corresponding to such class, and smaller values to the other output variables. The distribution of values among the outputs for a record provides information on how the large model represents knowledge. Therefore, the goal of economical deployment of a valid model can be achieved by training only the large model on the data, exploiting its better ability to learn concise knowledge representations, and then distilling such knowledge into the smaller model, by training it to learn the soft output of the large model [22]. When implementing a knowledge distillation system, it is important to consider its structure with three key components: knowledge, distillation algorithm, and teacher-student architecture [7]. A general teacher-student framework for knowledge distillation is shown in Figure 3.4.

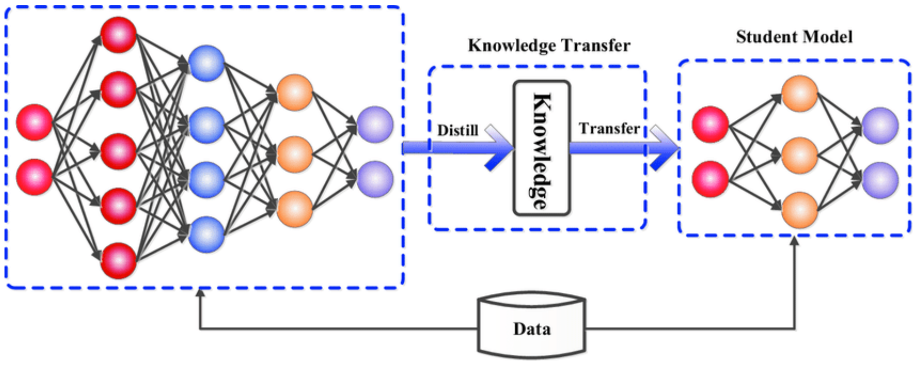


Figure 3.4: The generic teacher-student framework for knowledge distillation. [7]

For the purposes of this model, the teacher-student architecture consists of a CNN14 teacher model and a Random Forest classifier as the student model. The distillation algorithm would be based on class aggregation of hard targets given as output of the teacher model. Knowledge in this model would be the dataset of early media that was collected for this purpose. The following sections will describe each of the aforementioned terms, while the analysis of the early media dataset is reserved for a separate chapter.

3.3 Teacher model architecture

The teacher model used for this application is CNN14. CNN14 is based on conventional CNNs, which consist of several convolutional layers that contain several kernels that are convolved with the input feature maps to capture their local patterns [2]. CNNs adopted for audio tagging often use log mel spectrograms as input. Short-time Fourier transforms (STFTs) are applied to time-domain waveforms to calculate spectrograms. Then, mel filter banks are applied to the spectrograms, followed by a logarithmic operation to extract log mel spectrograms. CNN14 consists of 6 convolutional layers. Each convolutional block consists of 2 convolutional layers with a kernel size of 3×3. Batch normalization is applied between each convolutional layer, and the ReLU nonlinearity is used to speed up and stabilize the training. Average pooling of size 2×2 is applied to each convolutional block for downsampling, as 2×2 average pooling has been shown to outperform 2×2 max pooling. Global pooling is applied after the last convolutional layer to summarize the feature maps into a fixed-length vector. For global pooling, CNN14 uses the sum of averaged and maximized vectors. An extra fully-connected layer is added to the fixed-length vectors to extract embedding features, which can further increase their representation ability. For a particular audio pattern recognition task, a linear classifier is applied to the embedding features, followed by either a softmax nonlinearity for classification tasks or a sigmoid nonlinearity for tagging tasks. Dropout is applied after each downsampling operation and fully connected layers to prevent systems from overfitting. For training, the entire audio clip is used without cutting it into segments [2].

Table 3.1 summarizes the proposed CNN14 systems. The number after the “@” symbol indicates the number of feature maps. MP is the abbreviation of max pooling. The “Pooling 2×2” in 3.1 is average pooling with a size of 2×2.

CNN14
Log-mel spectrogram 1000 frames $\times$ 64 mel bins
(3×3 @ 64; BN, ReLU) $\times$ 2
Pooling 2×2
(3×3 @ 128; BN, ReLU) $\times$ 2
Pooling 2×2
(3×3 @ 256; BN, ReLU) $\times$ 2
Pooling 2×2
(3×3 @ 512; BN, ReLU) $\times$ 2
Pooling 2×2
(3×3 @ 1024; BN, ReLU) $\times$ 2
Pooling 2×2
(3×3 @ 2048; BN, ReLU) $\times$ 2
Global pooling
FC 2048, ReLU
FC 527, Sigmoid

Table 3.1: Architecture of CNN14 (Adapted from [2])

Let the waveform of an audio clip be denoted as x_n, where n is the index of audio clips, and $f(x_n) \in [0,1]^K$ is the output of CNN14 representing the presence probabilities of K sounds classes. The label of x_n is denoted as $y_n \in \{0,1\}^K$. A binary cross-entropy loss function l is used to train CNN14:

$$l = -\sum_{n=1}^{N} \left(y_n \cdot \ln f(x_n) + (1 - y_n) \cdot \ln(1 - f(x_n))\right), \qquad (3.1)$$

Where N is the number of training clips in AudioSet. In training, the parameters of $f(\cdot)$ are optimized by using gradient descent methods to minimize the loss function l [2].

3.4 Random Forest Classifier

The chosen student model in this study is a random forest classifier. Random forests or random decision forests are an ensemble learning method for classification, regression, and other tasks that works by creating a multitude of decision trees during training [28]. For classification tasks, the output of the random forest is the class selected by the most trees. Random forests correct for decision trees' habit of overfitting to their training set [29]. In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e., have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

An illustration of training a Random Forest model is displayed in Figure 3.5. The training dataset (in this case, of 250 rows and 100 columns) is randomly sampled with replacement *n* times. Then, a decision tree is trained on each sample. Finally, for prediction, the results of all *n* trees are aggregated to produce a final decision.

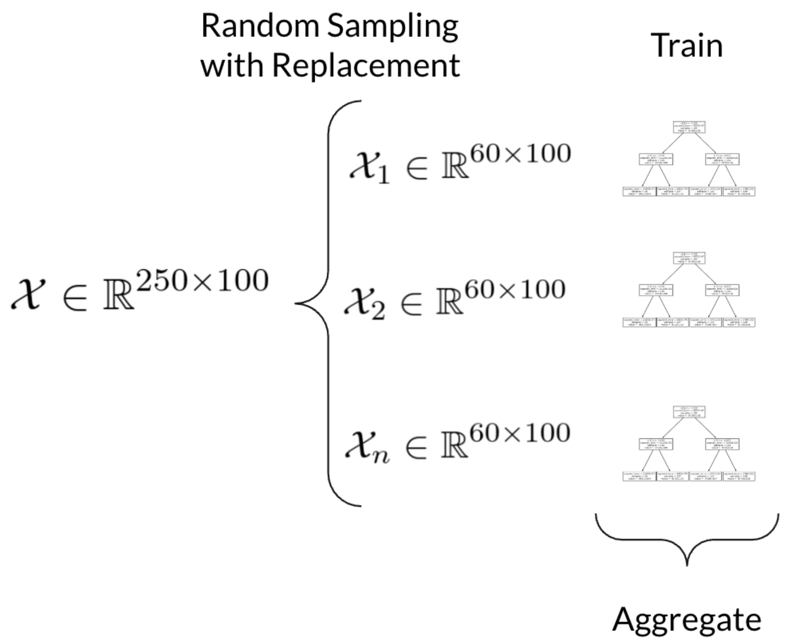

Random Sampling with Replacement
Train
$\mathcal{X} \in \mathbb{R}^{250\times100}$
$\mathcal{X}_1 \in \mathbb{R}^{60\times100}$
$\mathcal{X}_2 \in \mathbb{R}^{60\times100}$
$\mathcal{X}_n \in \mathbb{R}^{60\times100}$
Aggregate

Figure 3.5: Random forest training [8]

The benefit of a random forest classifier compared to other decision tree-based classifiers, such as gradient boosted trees, is that it performs well out of the box with minimal tuning [30].

3.5 Class aggregation

In order to be able to transfer the knowledge from the teacher model and make it suitable to be applied in real-time for early media classification, the original 527 classes that the teacher model classifies the audio into shall be mapped to one of 4 classes of interest: announcement, ringback, music, or silence. This mapping significantly reduces the complexity of the problem and therefore allows a simpler and more cost-efficient model to achieve high performance. Ambiguous mapping wouldn't be included in the dataset.

The output of a teacher model in knowledge distillation is a set of hard and soft targets. Hard targets represent the class the model inferred the input belongs to, while the soft targets represent the probabilities of belonging to each of the possible classes [22]. In the case of this model, only hard targets are going to be taken into consideration, since it would be difficult to incorporate such input attributes in the chosen student model and it wouldn't provide much value since during the teacher model testing it was observed that the resulting soft targets for the 4 classes tend to have low entropy.

Chapter 4

Datasets

This chapter is dedicated to the analysis of the datasets used to train the teacher and student models. Even though the teacher model training wasn't part of this project, it is still useful to analyze the composition of the dataset it was trained on.

4.1 AudioSet

As previously mentioned, the teacher model was trained on AudioSet[1]. AudioSet consists of an expanding ontology of 527 audio event classes and a collection of 2,084,320 human-labeled 10-second sound clips drawn from YouTube videos. The ontology is specified as a hierarchical graph of event categories, covering a wide range of human and animal sounds, musical instruments and genres, and common everyday environmental sounds. Due to the co-occurrence of labels, many classes have multiple examples.

To analyze the dataset in the context of early media classification, the relevant labels need to be mapped to one of the four categories of early media: speech, silence, music, and ringing. Ambiguous mappings weren't considered in the analysis.

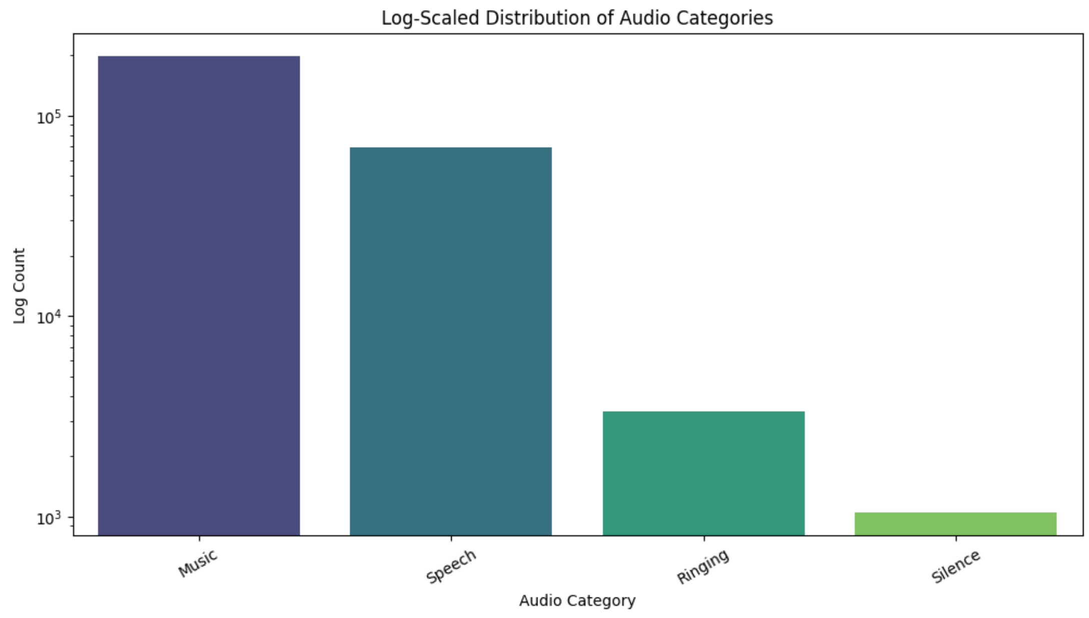


Figure 4.1: AudioSet class distribution

[1] https://research.google.com/audioset/index.html

From the visualized distribution in figure 4.1, it can be concluded that the music labels category is most represented in the dataset, while the silence label category is vastly underrepresented in the dataset.

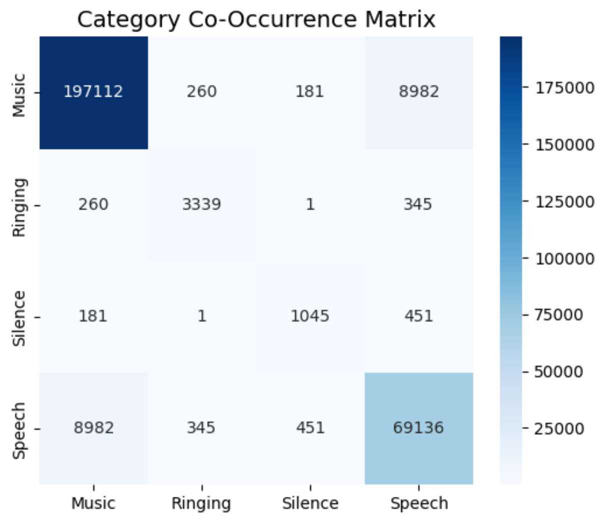


Figure 4.2: Co-occurence of audio event classes in AudioSet

From figure 4.2 it is visible that the data usually belongs to only one of the four label categories. It also shows that music is highly co-occurring with other categories, often appearing alongside speech.

AudioSet is balanced between the 527 sound classes it recognizes; however, the number of classes that are mapped to the 4 resulting classes needed for this model is unevenly distributed, which is visible from the previous figures. This could pose an issue as the teacher model might overfit to the music class and underfit for the silence class. This issue could be more prominent due to ambiguous mappings being removed from consideration.

4.2 Early Media dataset

The Early Media dataset used to train the student model is a collection of audio files that contain early media audio. Unlike AudioSet, which already contained the information necessary to perform the analysis of the dataset, this dataset needs significant preprocessing before a meaningful analysis can be performed.

The initial dataset contained 10,000 audio files of varied duration. To acquire the classes the audio belongs to, it needs to be batch labeled by the teacher model. For that purpose, each audio file is split into 1-second segments. Care was taken that all segments from one audio file belong to either the train or test split. Initially, the dataset was highly imbalanced, with 80% of audio segments belonging to the speech class. For that reason, class imbalance removal was performed. This procedure was done by removing a significant number of audio segments labeled with speech. The criteria for removal were the classification confidence, with segments belonging to the speech class with confidence lower than 0.899 being removed from the dataset. This procedure removed almost 70% of speech segments from the dataset. In total, there were 31,839 seconds of audio present in the dataset.

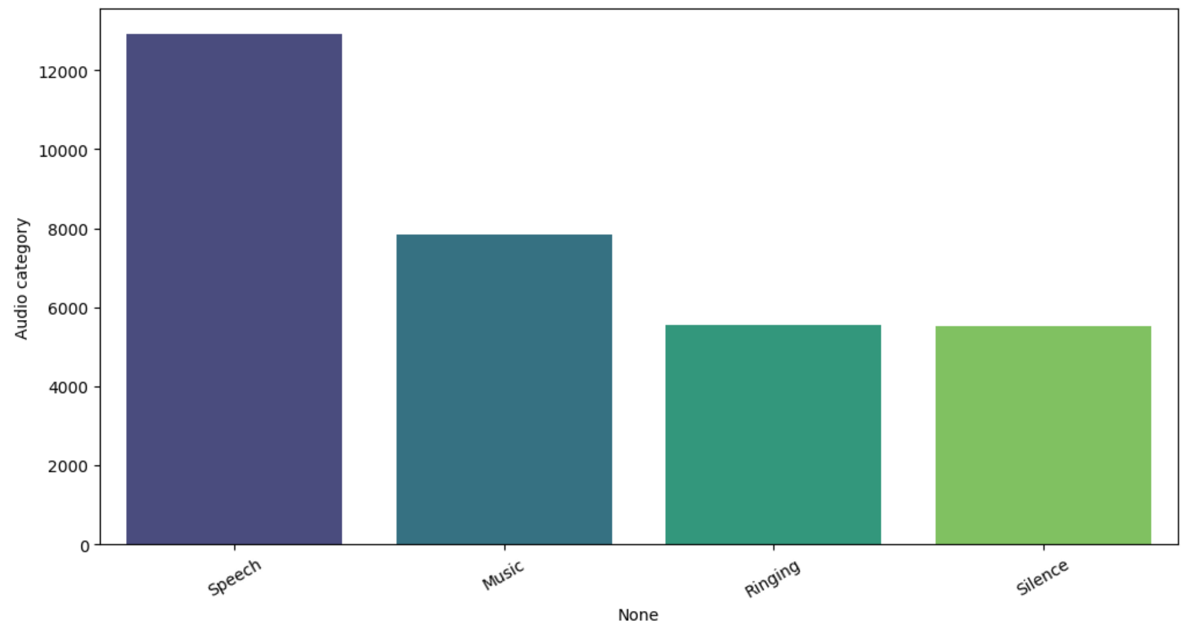


Figure 4.3: Early Media dataset class distribution

As visible from 4.3, the actual early media dataset has a more balanced distribution of data points for each class. Speech in this case is most prevalent, and that is the category that is most relevant, as it should be the target of audio fingerprinting after the classification is performed.

Considering that soft targets weren't taken into consideration, it is also useful to analyze their distribution to see what impact they could have on the training of the student model. Soft targets in this case are the probabilities for the audio to belong to each of the classes.

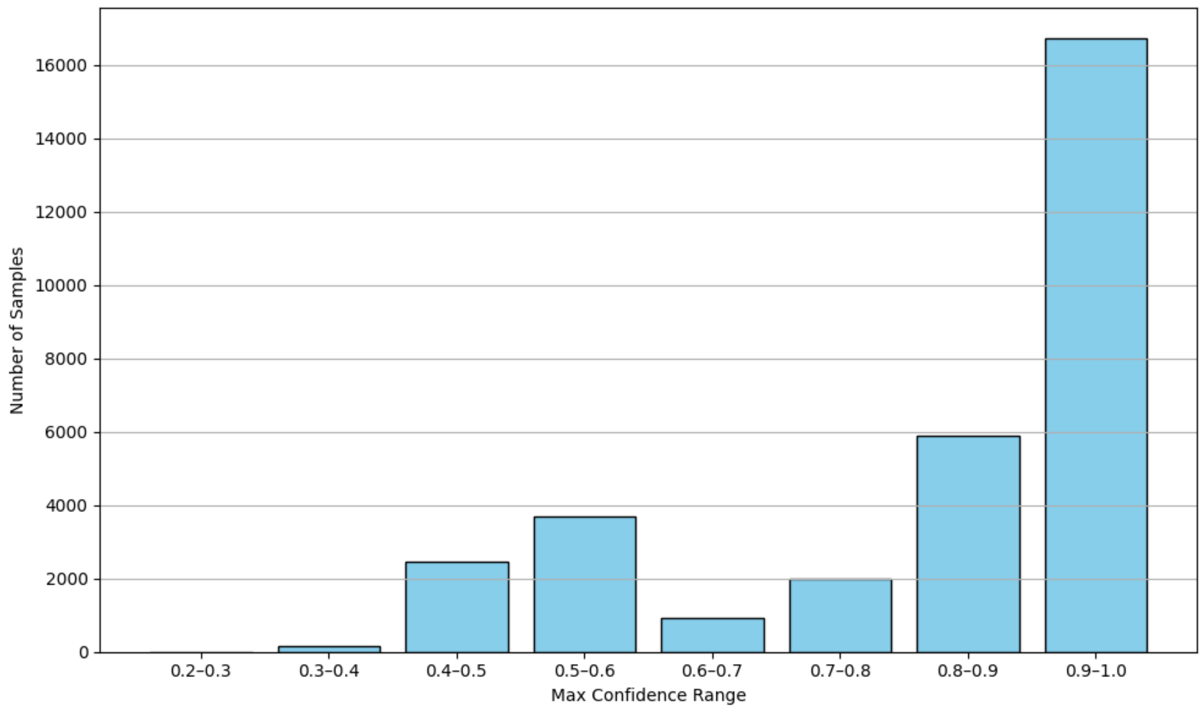


Figure 4.4: Sample counts by confidence intervals for the highest probability soft targets per sample

As visible from figure 4.4, the greatest amount of maximum probability soft targets belong in the 0.8-1.0 range, which suggests that overall there isn’t a lot of overlap between the classes.

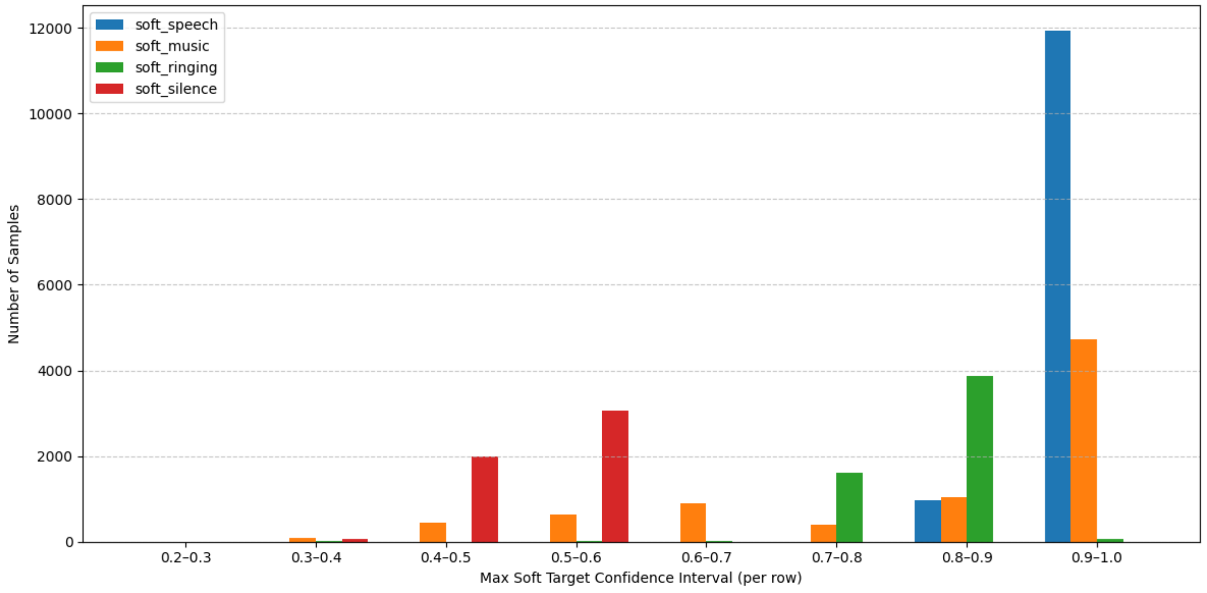


Figure 4.5: Sample counts by confidence intervals for the highest probability soft targets per sample per class

Figure 4.5 shows the distribution of sample counts by confidence intervals for the highest probability soft targets per sample per class. It is visible that there is a significant variation in the soft targets of each class. Speech is most prevalent when the probability of it being classified is between 0.9 and 1.0, which is expected due to the performed class balancing. Music is to a lesser extent prevalent in the 0.9-1.0 range. Other ranges are significantly smaller, thus less relevant. One important notice is that silence is most prevalent in the 0.4-0.6 range, suggesting that silence is classified with the least confidence of all of the classes.

It can be concluded that the poor distribution of AudioSet did not affect the classification of the early media dataset, as it had much more equally distributed samples between the classes. The low confidence of silence classification should be taken into consideration for possible false classifications of that class.

Chapter 5

Implementation

This chapter is dedicated to explaining the steps performed during the implementation of the proposed solution. The conclusion of this chapter includes the evaluation of the model with additional remarks on the possible implementation variations. It should be noted that to ensure best practices and development efficiency, parts of the implementation code were created with the help of AI-based coding tools.

5.1 Data preprocessing

Considering that the collected audio files do not contain any metadata, significant preprocessing was needed for the data to be used in the teacher and student models. Thankfully, the `librosa` library contains useful functions for this purpose.

Each file is loaded from a Google Drive folder, and it is added to a new array of audio in a different presentation. Namely, the audio gets added in the array as a floating-point time series, so it can be used as input in the teacher model. The CNN14 teacher model expects the sample rate of the input audio to be 32 kHz, which is also performed for each file with the `librosa.load` function.

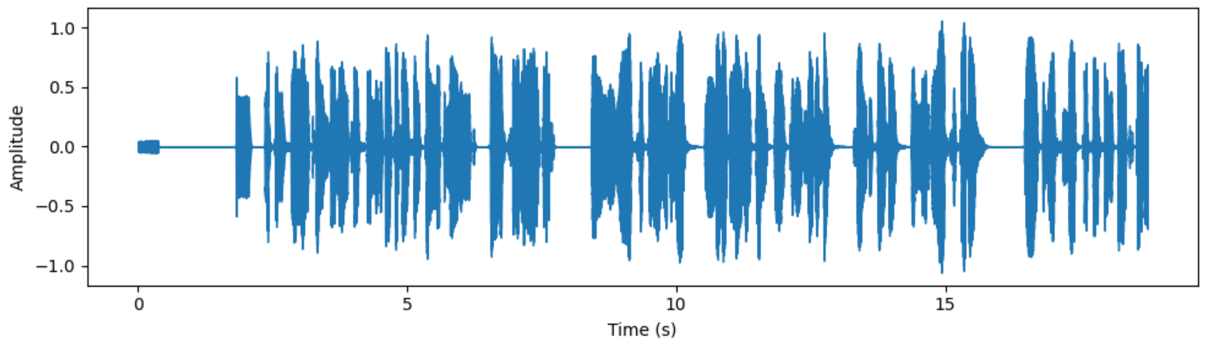


Figure 5.1: Waveform of early media audio

Figure 5.1 represents an example of one of these preprocessed audio files. It is a waveform of early media audio. The content of the audio belongs to the speech class. It contains the following announcement: "The person you are trying to reach is unable to take your call at the moment. Please try later". The same announcement is repeated in Hindi afterwards.

The floating point time series is converted to a Pytorch tensor, after which it can be successfully used as input for the teacher model.

5.2 Class aggregation

Another required step before acquiring the ground truth (hard target) from the teacher model is to implement class aggregation of the classes that the teacher model returns as the ground truth to the classes in which the student model performs the classification. This is achieved using a utility function that is called on each classification of the teacher model to perform the mapping. For ease of explanation, the classes that the teacher model classifies will be called *teacher classes*, while the four classes that the student model should classify into will be called *student classes*. Due to the ontology of AudioSet not being adapted for the student classes, the mapping was performed by manually adding each of the 527 teacher classes that were estimated to belong to one of the student classes into an array of mapping teacher classes.

As previously mentioned, there are a total of 527 teacher classes. Many of these can't be unambiguously mapped into student classes. Therefore, these teacher classes aren't mapped to any student class, which means that some teacher classes aren't taken into consideration by the student model.

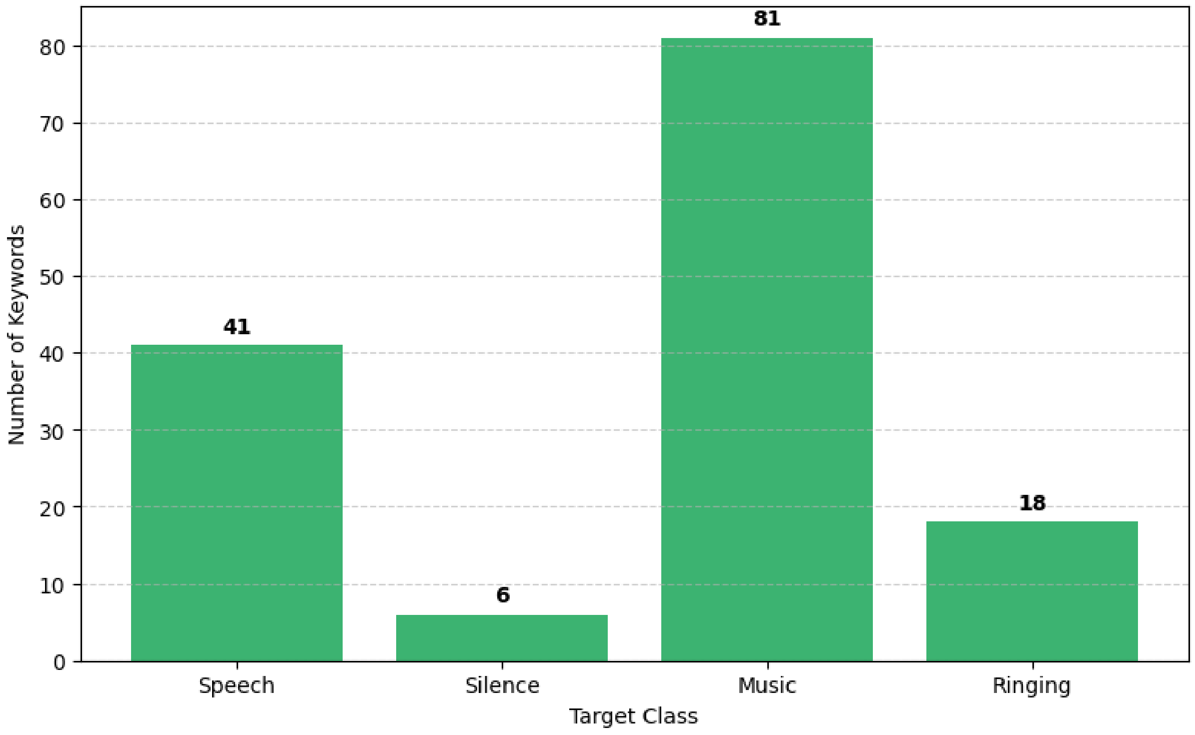


Figure 5.2: Number of mapped teacher classes into each student class

Figure 5.2 shows the number of teacher classes mapped to each student class. It can be concluded that 381 teacher classes are left unmapped. This number seems daunting; however, procedures during the preprocessing of the student model were made to accommodate the large number of unmapped teacher classes. Those procedures will be explained in the following sections.

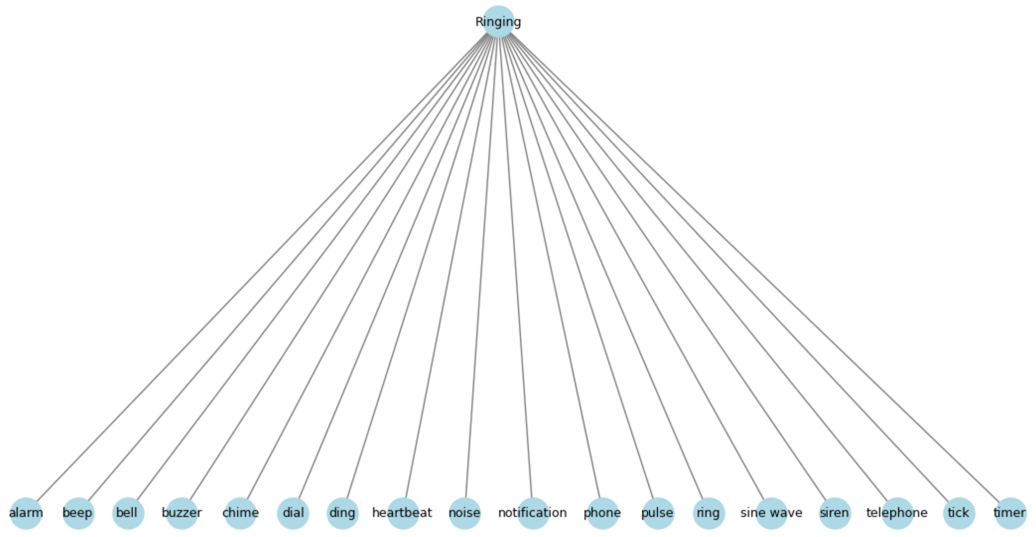


Figure 5.3: Class aggregation tree for the Ringing student class

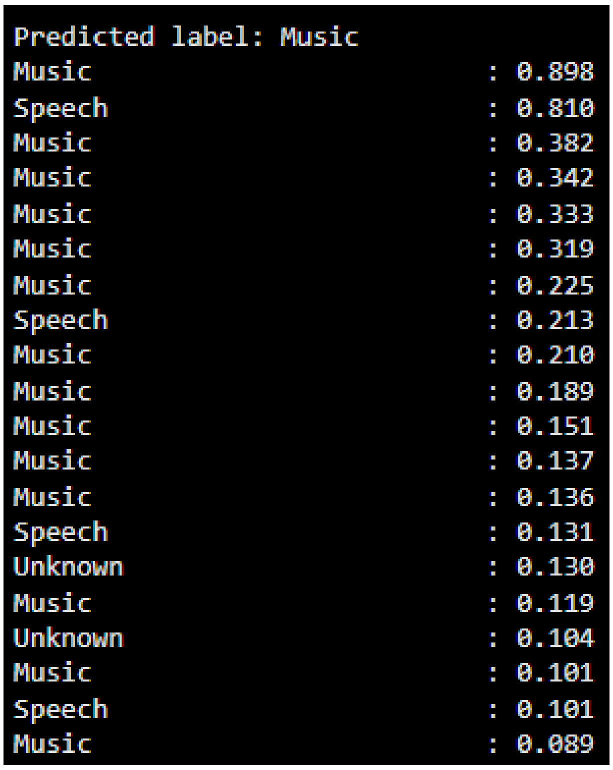
Figure 5.3 shows one of these mappings for the ringing student class. Some teacher classes, like heartbeat, initially didn't appear to fit into any of the student classes. In those cases, audio files labeled with those teacher classes were analyzed by listening to whether that teacher class would fit any of the classes. In the case of the heartbeat class, it was noticed that the audio labeled with that teacher's class contained a sound that was repeating up to a few times a second, which could be interpreted as a ringing tone. Therefore, that teacher's class was mapped to the ringing student class.

5.3 Teacher model inference

It is useful to mention the relevant implementation details of the teacher model before proceeding to explain its usage in this solution. Due to the specifics and generic capabilities of CNN14, various modifications were necessary to make it suitable for training a student model with real-time inference capabilities. CNN14 performs the labeling of each 10-millisecond segment of the audio file, which it receives as input [2]. This degree of granularity isn't necessary for this solution, as it increases the inference time needed to process an audio file.

In order to mitigate this issue, labeling was aggregated to one-second windows. If an aggregated second contains differently classified 10 millisecond segments, then that second will be removed from the teacher output dataset. This reduces the number of data points returned by the teacher model by a factor of 100. This solution also mitigates potential inconsistencies in the returned classes for such small segments as 10 milliseconds [1]. For example, there could be a situation where many seconds of speech could be followed by 10 milliseconds of ringing noise, followed by multiple seconds of speech.

Another challenge encountered with the generic nature of CNN14 is that the model performs multi-label classification. This means that the sum of probabilities of an input belonging to a specific class is not 1; rather, each class's probability is independent of the probability of other classes. This poses an issue when the teacher classes are mapped to student classes. Namely, in that case, a situation can exist where the highest probability teacher class is mapped to one student class, for example speech, while there can be multiple teacher classes mapped to some other student class that have only a slightly smaller probability of classification.



```
Predicted label: Music
Music                          : 0.898
Speech                         : 0.810
Music                          : 0.382
Music                          : 0.342
Music                          : 0.333
Music                          : 0.319
Music                          : 0.225
Speech                         : 0.213
Music                          : 0.210
Music                          : 0.189
Music                          : 0.151
Music                          : 0.137
Music                          : 0.136
Speech                         : 0.131
Unknown                        : 0.130
Music                          : 0.119
Unknown                        : 0.104
Music                          : 0.101
Speech                         : 0.101
Music                          : 0.089
```

Figure 5.4: Multi-label classification of the teacher model after class aggregation

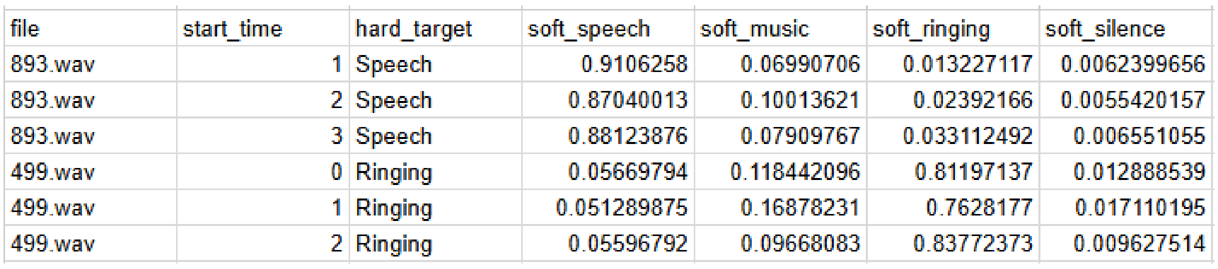
The described situation is visible from figure 5.4. Music is the most prevalent class, but only by a small margin ahead of speech. One attempted way to mitigate such an issue was to determine the final class based on a sum of a fixed amount of the highest probabilities. Even though in theory this method should improve the reliability of the data, it often resulted in favoring the lower probability classes while ignoring the first or second highest probabilities, which often resulted in wrong classifications. Thus, the most reliable method that was chosen in the implementation was to use the mapped class with the highest probability as the final class for that audio segment.

The teacher CNN14 model inference is the next step in building this pipeline. The PyTorch implementation of the model is available from a GitHub repository[1]. The model is then initialized using the default parameters suggested by its authors. Afterwards, the weights acquired from the same repository as the model are loaded. The model is then put into evaluation mode to ignore the configuration specific to training. With this procedure finished, the model is ready to perform inference. A loop goes over each audio file in the dataset. It transforms it into a floating-point array and performs teacher model inference on it. Then the one-second window aggregation is performed as already described. Afterwards, class aggregation is performed with care being taken to remove the one-second audio files whose 10 millisecond segments aren't consistent. For evaluation purposes, normalized soft targets are also calculated.

[1]`https://github.com/qiuqiangkong/audioset_tagging_cnn`

5.4 Student model data preprocessing

The starting point of student model training is the .csv dataset created with teacher model inference. Sample datapoints from the dataset are visible on figure 5.5



file	start_time	hard_target	soft_speech	soft_music	soft_ringing	soft_silence
893.wav	1	Speech	0.9106258	0.06990706	0.013227117	0.0062399656
893.wav	2	Speech	0.87040013	0.10013621	0.02392166	0.0055420157
893.wav	3	Speech	0.88123876	0.07909767	0.033112492	0.006551055
499.wav	0	Ringing	0.05669794	0.118442096	0.81197137	0.012888539
499.wav	1	Ringing	0.051289875	0.16878231	0.7628177	0.017110195
499.wav	2	Ringing	0.05596792	0.09668083	0.83772373	0.009627514

Figure 5.5: Student model training dataset

5.5 Student model training

To train the student model, several steps needed to be performed that are performed during teacher model inference by the CNN14 model itself. The key step is to represent audio in a form suitable for feature extraction. This is performed by representing the audio as a mel-frequency spectrogram (mel spectrogram). In short, a spectrogram will represent a signal's amplitude as it varies over time at different frequencies. The mel-frequency spectrogram is a variation of a spectrogram whose frequencies are converted to a mel scale. This scale is used to make the distances between pitches sound equally distant to the listener. This is performed because humans perceive differences in sound pitch differently based on the frequency of the sound, not just the absolute difference in the frequency.

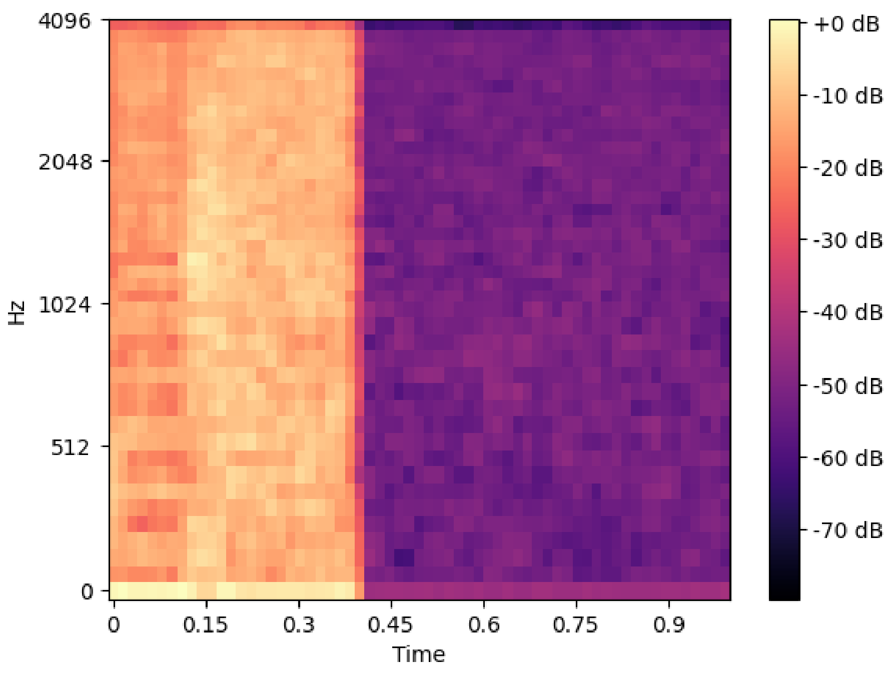


Figure 5.6: Mel frequency sprectrogram of one second of speech labeled audio used for student model traning. Amplitude is on a logarithmic scale

For training the student model, the student model training dataset is split into an 80-20 train-test split. As previously mentioned, care was taken that all 1-second clips belonging to one file would belong fully to either the train or test split. Afterwards, each 1-second audio clip is represented with a mel spectrogram feature vector. This way, the data is fully prepared for training with a random forest classifier.

The model is trained with parameters for a random forest classifier that include 500 estimator trees and a random seed of 42. Detailed model evaluation is provided in the following chapter.

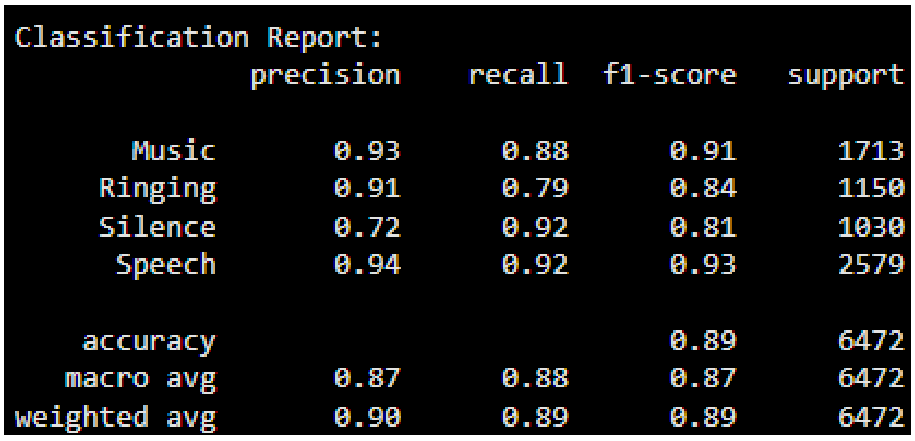
Chapter 6

Model evaluation

Several performance metrics are relevant for this model. Most important of these is the inference time and the accuracy of the trained model. As previously mentioned, 20% of the early media dataset is used for testing. Firstly, a classification report will be revealed regarding the testing of the model. This serves to display the general performance metrics of the modal. Then, a confusion matrix will be introduced for a detailed analysis of the accuracy of the model in order to pinpoint which pairs of classes are most mistakenly classified. Finally, an experiment will be performed to compare the performance of the student model against the teacher.

6.1 Classification report

The initial classification report is visible on figure 6.1



```
Classification Report:
              precision    recall  f1-score   support

       Music       0.93      0.88      0.91      1713
     Ringing       0.91      0.79      0.84      1150
     Silence       0.72      0.92      0.81      1030
      Speech       0.94      0.92      0.93      2579

    accuracy                           0.89      6472
   macro avg       0.87      0.88      0.87      6472
weighted avg       0.90      0.89      0.89      6472
```

Figure 6.1: Initial classification report

The model shows good results with an average accuracy of 0.89. All classes show good results with precision over 0.9, except for the silence class, whose precision is only 0.72, suggesting it often gets confused with other classes. This is not surprising since the teacher model classified the silence with the least confidence. Recall and F1-score also show solid results, which make the case for the model being usable in practical applications.

6.2 Confusion matrix

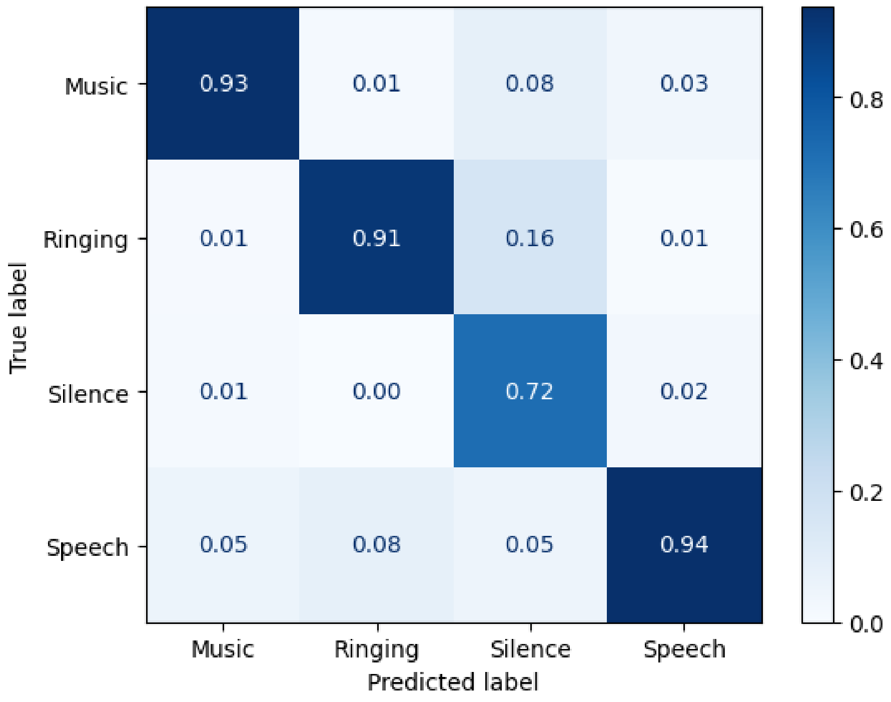


Figure 6.2: Column-normalized confusion matrix

As seen from the confusion matrix in Figure 6.2, the model shows great results in all but the silence prediction. The main issue is that the model over-predicts the silence class, most probably caused by the low confidence for the silence class, which leads to ambiguity. Even though this silence prediction seems worrying, further comparison to the student model will reveal that this performance is an inherent limitation of the chosen early media classes.

6.3 Student-teacher model comparison

An important evaluation of the student model regards its performance against the teacher. As previously mentioned, the main goal of the student model is to perform the inference in a significantly shorter time frame. All the while, the model should minimize the loss of precision against the teacher model. For this comparison, an experiment was performed using 1,000 1-second audio segments from the test split. The segments were used to compare the inference time and accuracy between CNN14 and the trained random forest classifier. It should be evident that the inference time of the student model is lower, while preserving the accuracy of the teacher model. The experiment was performed on the default Google Colab's CPU architecture of Intel(R) Xeon(R) CPU @ 2.20GHz.

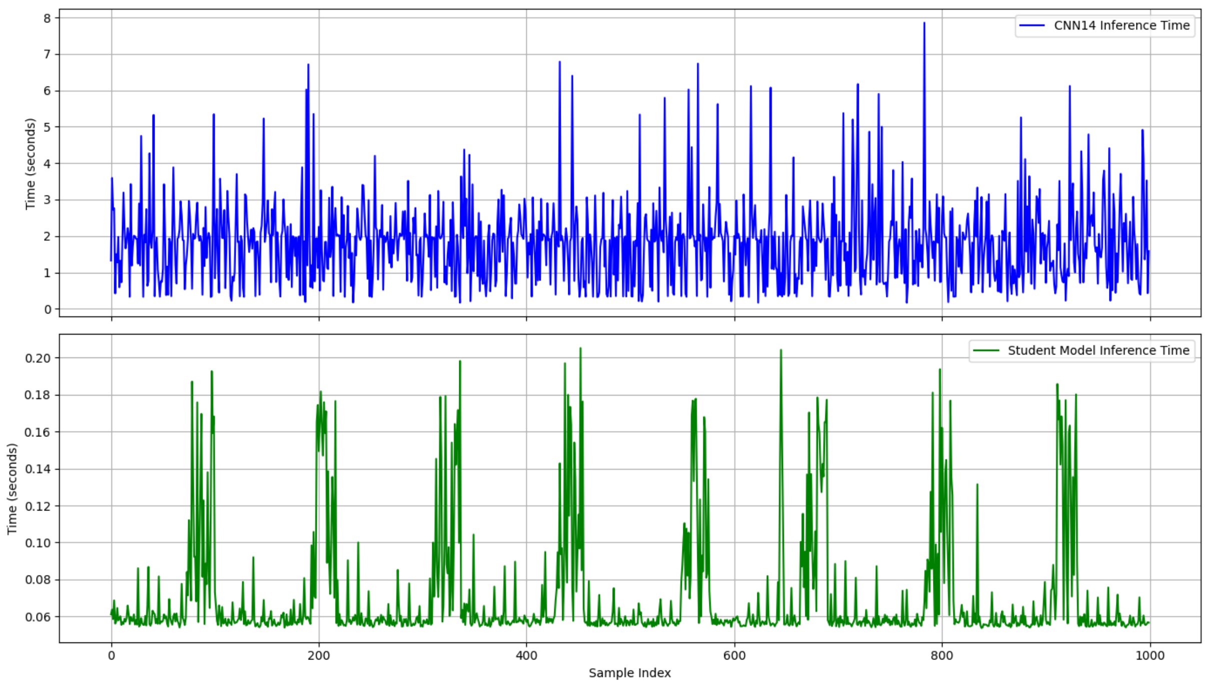
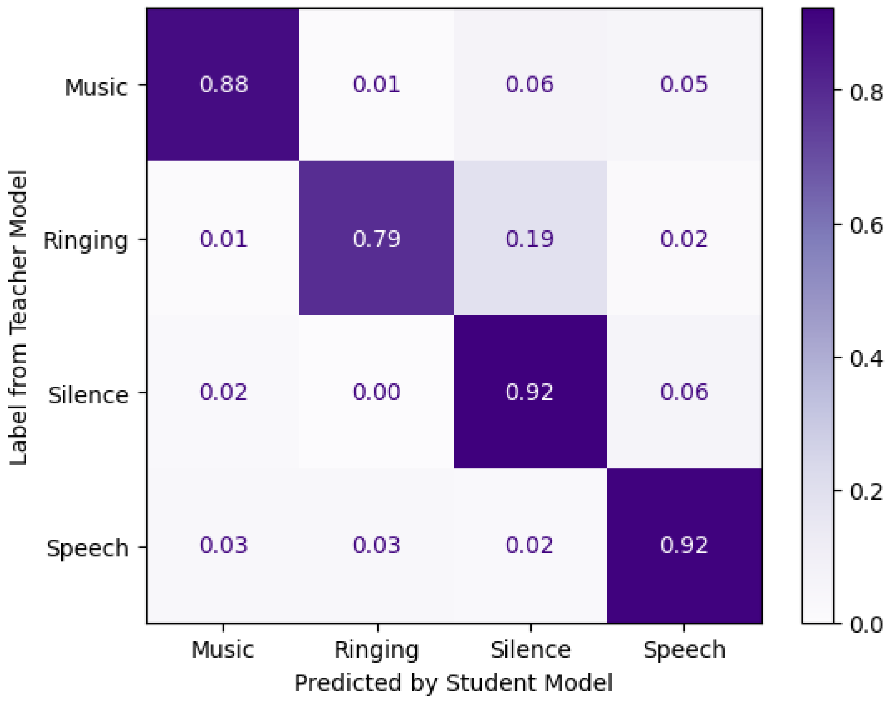


Figure 6.3: Comparison of inference time of the teacher and student models

Figure 6.3 shows the inference time difference between the student and teacher models. There is significant variance in the inference time; however, the focal point of the comparison is the average inference time between the models. On Google Colab's CPU runtime, the student model averages an inference time of 71 ms, while the teacher model takes on average 1813.2 ms to perform the inference on the same hardware. This is an improvement by a factor of 25.5.



Label from Teacher Model \ Predicted by Student Model	Music	Ringing	Silence	Speech
Music	0.88	0.01	0.06	0.05
Ringing	0.01	0.79	0.19	0.02
Silence	0.02	0.00	0.92	0.06
Speech	0.03	0.03	0.02	0.92

Figure 6.4: Confusion matrix for the student and teacher models

Figure 6.4 shows the comparison of early media class predictions from the teacher and student models. As is visible, all classes show a reliable level of accuracy from the student model. The ringing class is lacking, most likely due to it being overrepresented in AudioSet, as it seems to often be confused with the low-confidence silence class. Most surprisingly, the silence class showed the least amount of loss of precision compared to the teacher model. This is probably caused by the fact that the silence class was vastly underrepresented in AudioSet. Also, the silence class was proportionately more present in the early media dataset.

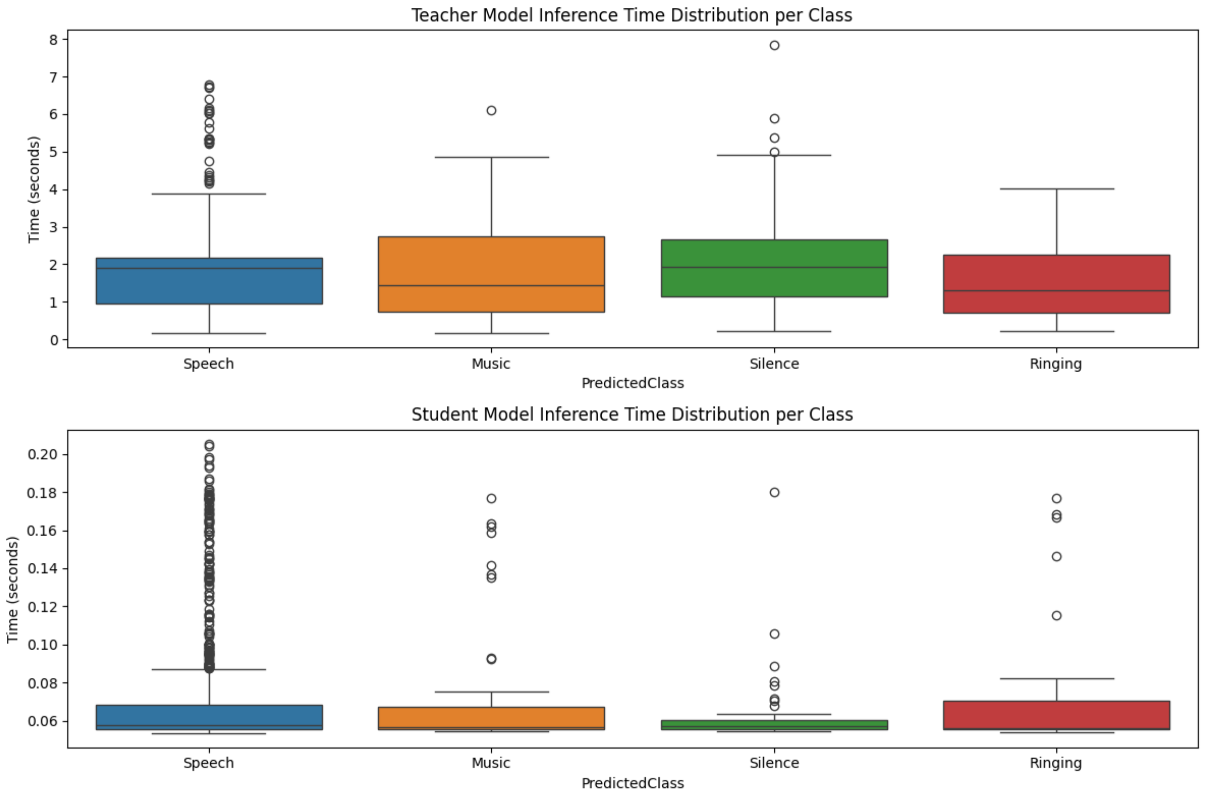


Figure 6.5: Comparison of inference time distribution per class for CNN14 and the student model

Figure 6.5 shows the distribution of inference time for the two models for the four early media classes. The two models have a similar distribution. One notable observation is that the student model has more outliers in the speech class; however, this fact doesn't affect the overall model performance. It is also visible that apart from the already mentioned outliers of the speech class, no major difference in inference time between classes was observed for both models.

6.4 Applicability to industry constraints

Once a VoIP call is established, if any early media is exchanged, it would be commonly stored in a buffer [1]. When 1 second of audio has been accumulated into the buffer, the content from the buffer is used to compute the mel-spectrogram. The mel-spectrogram is then used by the model and classified into one of four possible classes. In most cases, the output of the model can be treated as the final result. However, if a higher confidence is required, the result can be memorized, and the model can be invoked again when additional data arrives. Additional waiting may be very useful for eliminating cases such as false positives when detecting silence. In an industrial setting, the model's classification could be considered final once it achieves a set number of equal classifications. The model thus exhibits a stream-like behaviour, enabling processing of early media of various and a priori unknown duration in real-time. This can be done continuously, in increments of 1 second, without waiting for the entire file to be received [1].

Chapter 7

Conclusion

This thesis tackled the problem of time-constrained audio classification of early media in voice communication. For achieving a higher degree of accuracy, knowledge distillation was utilized from a resource-demanding neural network. The trained student model, a random forest classifier, managed to preserve a substantial amount of precision from the teacher neural network, whilst drastically reducing inference time.

The implemented solution proved capable of classifying the early media audio into the 4 classes. This model can now be used in voice communications as a classifier of early media in real-time.

7.1 Future Work

There are many potential improvements to the existing model. One such relates to the chosen teacher model. CNN14 was, until recently, the state-of-the-art in regards to audio classification in terms of the mAP metric. However, newer, foundation models such as CLAP have seen remarkable performance with a diverse set of inputs. Using such a foundation model as the teacher in this architecture might prove to be more capable than CNN14, considering the recognized disadvantages of CNN14 for this application.

Another possible modification is to use a different student model, such as a Gradient Boosted Trees model, similar to the approach in [1]. This approach is expected to increase the accuracy compared to the used random forest classifier, while having a slightly more complex architecture.

Another possible augmentation of the existing pipeline is to add an audio fingerprinting technique for speech content determination. This component would undoubtedly increase the processing time and complexity of the architecture. However, this component is crucial in pinpointing the root cause of call failure, as this technique can utilize previously recorded audio to draw conclusions about the call failure.

With this model implemented, a SIP response code mapping can be easily performed from the classes the model recognizes. This mapping depends on the reason the model is used. For example, if the goal is to detect the reason for call failure, a mapping such as the one in table 7.1 could be performed:

Early Media Class	**SIP Response Code**
Ringing	180 Ringing
Silence	408 Request timeout
Speech	403 Forbidden
Music	183 Session progress

Table 7.1: Example early media class to SIP response code mapping

Implementing audio fingerprinting for the speech class could further refine such a mapping and improve on the overall usability of the model.

Appendices

Appendix A

Source code

The source code is also available on GitHub[1].

```
!git clone https://github.com/qiuqiangkong/audioset_tagging_cnn
!apt install -y graphviz libgraphviz-dev
!pip install librosa soundfile wget torchlibrosa pygraphviz

%cd audioset_tagging_cnn/pytorch

import glob
import joblib
import librosa
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import os
import pandas as pd
import seaborn as sns
import soundfile as sf
import time
import torch
import torchaudio

from collections import defaultdict
from google.colab import drive
from pathlib import Path
from pytorch_utils import move_data_to_device
from models import Cnn14
from models import Cnn14_DecisionLevelMax
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report,
   confusion_matrix, ConfusionMatrixDisplay
from tqdm import tqdm
```

[1]https://github.com/FarukZahiragic/early-media-classification

```
drive.mount('/content/drive')

device = torch.device("cuda" if torch.cuda.is_available() else
   "cpu")

model = Cnn14_DecisionLevelMax(sample_rate=32000, window_size
   =1024, hop_size=320, mel_bins=64, fmin=50, fmax=14000,
   classes_num=527)

checkpoint = torch.load('/content/drive/MyDrive/bsc_thesis/
   Cnn14_mAP=0.431.pth', map_location=device)
model.load_state_dict(checkpoint['model'])
model.to(device)
model.eval()

with open('/content/drive/MyDrive/bsc_thesis/
   class_labels_indices.csv') as f:
    lines = f.readlines()[1:]
    idx_to_label = [line.strip().split(',')[2].strip('"') for
       line in lines]

audio_path = '/content/drive/MyDrive/bsc_thesis/
   early_media_sample/37.wav'
(audio, sr) = librosa.load(audio_path, sr=32000, mono=True)

audio_tensor = torch.tensor(audio).float().unsqueeze(0)  # (1,
   time_steps)
audio_tensor = move_data_to_device(audio_tensor, device)

with torch.no_grad():
    output = model(audio_tensor, None)
    clipwise_output = output['clipwise_output'].data.cpu().
       numpy()[0]

waveform, _ = librosa.load(audio_path, sr=32000)
plt.figure(figsize=(10, 3))
librosa.display.waveshow(waveform, sr=32000)

plt.xlabel("Time␣(s)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()

y, _ = librosa.load(audio_path, sr=32000, duration=1)
mel = librosa.feature.melspectrogram(y=y, sr=32000, n_mels=64)
logmel = librosa.power_to_db(mel)
librosa.display.specshow(logmel, sr=32000, x_axis='time',
   y_axis='mel')
```

```
plt.ylim(top=4096)
plt.colorbar(format='%+2.0f␣dB')
plt.show()

category_mapping = {
    "Speech": [
        "speech", "talking", "conversation", "narration", "
           monologue", "dialogue", "discussion",
        "babbling", "shout", "whispering", "chanting", "yell",
           "laugh", "cry", "bellow",
        "whoop", "scream", "giggle", "snicker", "chuckle", "
           whimper", "moan", "sigh",
        "mantra", "groan", "grunt", "breathing", "wheeze", "
           snooring", "gasp", "pant",
        "snort", "cough", "throat␣clearing", "sneeze", "sniff",
            "chewing", "biting", "gargling",
        "hiccup", "cheering", "speech", "cough"
    ],
    "Silence": [
        "silence", "quiet", "ambient", "run", "static", "
           vibration"
    ],
    "Music": [
        "music", "singing", "instrument", "orchestra", "piano",
            "guitar", "choir", "song",
        "humming", "melody", "drum", "beat", "rhythm", "
           synthesizer", "electronic",
        "yodeling", "chant", "rapping", "whistling", "applause"
           , "banjo",
        "sitar", "mandolin", "zither", "ukulele", "piano", "
           organ", "sampler", "harpsichord",
        "percussion", "timpani", "tabla", "cymbal", "hi-hat", "
           wood␣block", "tambourine",
        "maraca", "gong", "mallet", "marimba", "glockenspiel",
           "vibraphone", "steelpan",
        "orchestra", "horn", "trumpet", "trombone", "string", "
           violin", "pizzicato", "cello", "bass",
        "flute", "saxophone", "clarinet", "harp", "tuning␣fork"
           , "chime", "harmonica",
        "accordion", "bagpipes", "didgeridoo", "shofar", "
           theremin", "scratching",
        "heavy␣metal", "rock", "grunge", "rhythm", "reggae", "
           country", "bluegrass", "funk",
        "jazz", "disco", "opera", "techno", "dubstep", "
           electronica", "flamenco", "blues",
        "capella", "ska", "lullaby"
    ],
    "Ringing": [
        "ring", "bell", "phone", "alarm", "chime", "beep", "
           notification", "timer", "noise",
        "pulse", "sine␣wave", "heartbeat", "telephone", "dial",
            "siren", "buzzer", "tick", "ding"
    ]
}

def categorize_label(display_name):
    display_name_lower = display_name.lower()

    for category, keywords in category_mapping.items():
        if any(keyword in display_name_lower for keyword in
           keywords):
            return category

    return "Unknown"

plt.figure(figsize=(8, 5))
bars = plt.bar(counts.keys(), counts.values(), color='
   mediumseagreen')

for bar in bars:
    height = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        height + 1,
        str(height),
        ha='center',
        va='bottom',
        fontsize=10,
        fontweight='bold'
    )

plt.xlabel("Target␣Class")
plt.ylabel("Number␣of␣Keywords")
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

counts = {category: len(set(keywords)) for category, keywords
   in category_mapping.items()}

plt.figure(figsize=(8, 5))
plt.bar(counts.keys(), counts.values(), color='mediumseagreen')
plt.xlabel("Target␣Class")
plt.ylabel("Number␣of␣Keywords")
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

category_mapping = {
    "Ringing": [
        "ring", "bell", "phone", "alarm", "chime", "beep", "
           notification", "timer", "noise",
        "pulse", "sine␣wave", "heartbeat", "telephone", "dial",
            "siren", "buzzer", "tick", "ding"
    ]
}

G = nx.DiGraph()

for category, keywords in category_mapping.items():
    for keyword in sorted(set(keywords)):
        G.add_edge(category, keyword)

pos = nx.nx_agraph.graphviz_layout(G, prog='dot')

plt.figure(figsize=(12,6))
nx.draw(
    G, pos,
    with_labels=True,
    arrows=False,
    node_size=500,
    font_size=9,
    node_color='lightblue',
    edge_color='gray'
)

plt.axis('off')
plt.tight_layout()
plt.show()

label = predict_label_from_output(clipwise_output)
print(f"Predicted␣label:␣{label}")

top_k = 20
top_indices = np.argsort(clipwise_output)[::-1][:top_k]
for idx in top_indices:
  category = categorize_label(idx_to_label[idx])
  print(f"{category:<30}:␣{clipwise_output[idx]:.3f}")

AUDIO_DIR = "/content/drive/MyDrive/bsc_thesis/
   early_media_sample"
SAMPLE_RATE = 32000
HOP_SIZE = 320
FRAMES_PER_SECOND = SAMPLE_RATE // HOP_SIZE
```

```
wav_files = glob.glob(os.path.join(AUDIO_DIR, '*.wav'))

print(wav_files)

# Batch labelling for the whole dataset
index_to_category = {}
for idx, label in enumerate(idx_to_label):
    mapped = categorize_label(label)
    if mapped != "Unknown":
        index_to_category[idx] = mapped

target_categories = ["Speech", "Music", "Ringing", "Silence"]

results = []
audio_files = [f for f in os.listdir(AUDIO_DIR) if f.endswith('
   .wav')]

for fname in tqdm(audio_files, desc="Extracting␣normalized␣soft
   ␣targets"):
    fpath = os.path.join(AUDIO_DIR, fname)

    try:
        waveform, _ = librosa.load(fpath, sr=SAMPLE_RATE, mono=
           True)
        x = torch.tensor(waveform).float().unsqueeze(0)
        x = move_data_to_device(x, device)

        with torch.no_grad():
            output = model(x, None)
            print(f"Output␣keys␣for␣{fname}:␣{output.keys()}")
            frame_probs = output['framewise_output'].data.cpu()
               .numpy()[0]  # shape: (T, 527)

        total_frames = frame_probs.shape[0]
        num_seconds = total_frames // FRAMES_PER_SECOND

        for sec in range(num_seconds):
          start_idx = sec * FRAMES_PER_SECOND
          end_idx = start_idx + FRAMES_PER_SECOND
          second_slice = frame_probs[start_idx:end_idx]  #
             shape: (100, 527)

          # For each frame, determine top predicted class
          top_classes = np.argmax(second_slice, axis=1)

          # Map those to categories
          mapped_categories = [categorize_label(idx_to_label[c
             ]) for c in top_classes]

          if len(set(mapped_categories)) != 1:
              continue  # if mixed predictions then skip this
                 second

          category = mapped_categories[0]
          if category == "Unknown":
              continue

          # Aggregate soft targets from all frames
          category_probs = defaultdict(float)
          for frame in second_slice:
              for idx, prob in enumerate(frame):
                  cat = index_to_category.get(idx, None)
                  if cat:
                      category_probs[cat] += prob

          soft_vector = np.array([
              category_probs.get("Speech", 0.0),
              category_probs.get("Music", 0.0),
              category_probs.get("Ringing", 0.0),
              category_probs.get("Silence", 0.0)
          ])
          total = np.sum(soft_vector)
          if total == 0:
              continue
          normalized_soft = soft_vector / total

          row = {
              "file": fname,
              "start_time": sec,
              "hard_target": category,
              "soft_speech": normalized_soft[0],
              "soft_music": normalized_soft[1],
              "soft_ringing": normalized_soft[2],
              "soft_silence": normalized_soft[3],
          }
          results.append(row)
    except Exception as e:
        print(f"Error␣processing␣{fname}:␣{e}")

OUTPUT_CSV = "/content/drive/MyDrive/bsc_thesis/audio_labels.
   csv"

df = pd.DataFrame(results)
df.to_csv(OUTPUT_CSV, index=False)
```

```
print(f"\nSaved␣results␣to:␣{OUTPUT_CSV}")

df = df[~((df["hard_target"] == "Speech") & (df["soft_speech"]
    <= 0.899))]

category_counts = df["hard_target"].str.get_dummies(sep=",").
    sum().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(x=category_counts.index, y=category_counts.values,
    hue=category_counts.index, palette="viridis", legend=False)

plt.yscale("linear")
plt.ylabel("Count")
plt.ylabel("Audio␣category")
plt.xticks(rotation=30)
plt.show()

soft_target_columns = ['soft_speech', 'soft_music', '
    soft_ringing', 'soft_silence']

df['max_confidence'] = df[soft_target_columns].max(axis=1)

bins = np.arange(0.2, 1.01, 0.1)  # Bin edges
labels = [f"{round(bins[i],2)}-{round(bins[i+1],2)}" for i in
    range(len(bins)-1)]

df['confidence_bin'] = pd.cut(df['max_confidence'], bins=bins,
    labels=labels, right=False)

bin_counts = df['confidence_bin'].value_counts().sort_index()

plt.figure(figsize=(10, 6))
plt.bar(bin_counts.index.astype(str), bin_counts.values, color=
    'skyblue', edgecolor='black')
plt.xlabel('Max␣Confidence␣Range')
plt.ylabel('Number␣of␣Samples')
plt.grid(axis='y')
plt.tight_layout()
plt.show()

le = LabelEncoder()
df["label_encoded"] = le.fit_transform(df["hard_target"])

def extract_logmel_features(file_path, sr=32000, n_mels=64,
    duration=1.0):
     try:
        y, _ = librosa.load(file_path, sr=sr, duration=duration
           )
        if len(y) < int(sr * duration):
            y = np.pad(y, (0, int(sr * duration) - len(y)))
        mel_spec = librosa.feature.melspectrogram(y=y, sr=sr,
           n_mels=n_mels)
        logmel_spec = librosa.power_to_db(mel_spec)
        return logmel_spec.mean(axis=1)  # Average over time
           axis
    except Exception as e:
        print(f"Error␣processing␣{file_path}:␣{e}")
        return None

unique_clips = df["file"].unique()
train_clips, test_clips = train_test_split(
    unique_clips,
    test_size=0.2,
    stratify=df.groupby("file")["hard_target"].first(),
    random_state=42
)

train_df = df[df["file"].isin(train_clips)]
test_df = df[df["file"].isin(test_clips)]

def extract_features_and_labels(df_subset):
    features, labels = [], []
    for _, row in df_subset.iterrows():
        feat = extract_logmel_features(os.path.join(AUDIO_DIR,
           row["file"]))
        if feat is not None:
            features.append(feat)
            labels.append(row["label_encoded"])
    return np.array(features), np.array(labels)

X_train, y_train = extract_features_and_labels(train_df)
X_test, y_test = extract_features_and_labels(test_df)

clf = RandomForestClassifier(n_estimators=500, random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
print("Classification␣Report:")
print(classification_report(y_test, y_pred, target_names=le.
   classes_))

cm_raw = confusion_matrix(y_test, y_pred)

cm_col_norm = cm_raw / cm_raw.sum(axis=0, keepdims=True)
cm_col_norm = np.nan_to_num(cm_col_norm)

disp = ConfusionMatrixDisplay(confusion_matrix=cm_col_norm,
   display_labels=le.classes_)
disp.plot(cmap="Blues", values_format=".2f")

plt.show()

cm_normalized = confusion_matrix(y_test, y_pred, normalize='
   true')

disp = ConfusionMatrixDisplay(confusion_matrix=cm_normalized,
   display_labels=le.classes_)
disp.plot(cmap="Purples", values_format=".2f")

plt.xlabel("Predicted␣by␣Student␣Model")
plt.ylabel("Label␣from␣Teacher␣Model")
plt.show()

joblib.dump(clf, "rf_student_model.pkl")
joblib.dump(le, "label_encoder.pkl")

def predict(filepath):
    feat = extract_logmel_features(filepath)
    if feat is None:
        return None
    feat = feat.reshape(1, -1)
    pred_encoded = clf.predict(feat)[0]
    return le.inverse_transform([pred_encoded])[0]

test_files = [os.path.join(AUDIO_DIR, "100.wav"),
              os.path.join(AUDIO_DIR, "101.wav"),
              os.path.join(AUDIO_DIR, "102.wav"),
              os.path.join(AUDIO_DIR, "102.wav"),
              os.path.join(AUDIO_DIR, "110.wav"),
              os.path.join(AUDIO_DIR, "111.wav")]

for file in test_files:
    pred = predict(file)
    print(f"{file}:␣{pred}")

teacher_times = []
AUDIO_SEGMENTS = test_df["file"].unique()[:1000]

for fname in AUDIO_SEGMENTS:
    path = os.path.join(AUDIO_DIR, fname)
    waveform, _ = librosa.load(path, sr=32000, mono=True)
    x = torch.tensor(waveform).float().unsqueeze(0).to(device)

    with torch.no_grad():
        start = time.time()
        _ = model(x, None)  # teacher inference
        end = time.time()

    teacher_times.append(end - start)

avg_teacher_time = np.mean(teacher_times)
print(f"Average␣inference␣time␣per␣audio␣file␣(Teacher):␣{
   avg_teacher_time:.4f}␣seconds")

predicted_classes = []
student_times = []

for fname in AUDIO_SEGMENTS:
    path = os.path.join(AUDIO_DIR, fname)

    start = time.time()

    feat = extract_logmel_features(path)

    if feat is not None:
        # Prediction
        _ = clf.predict([feat])

    end = time.time()
    prediction = clf.predict([feat])[0]
    predicted_label = le.inverse_transform([prediction])[0]
    predicted_classes.append(predicted_label)
    student_times.append(end - start)

avg_student_time = np.mean(student_times)
print(f"Corrected␣avg␣inference␣time␣per␣sample␣(Student):␣{
   avg_student_time␣*␣1000:.2f}␣ms")

models = ["Teacher␣(CNN14)", "Student␣(Random␣Forest)"]
times = [avg_teacher_time, avg_student_time]

times_ms = [t * 1000 for t in times]

plt.figure(figsize=(6, 4))
bars = plt.bar(models, times_ms, color=["#6a5acd", "#2e8b57"])

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height + 1, f"{
       height:.2f}␣ms",
             ha='center', va='bottom', fontsize=10)

plt.ylabel("Average␣Inference␣Time␣(ms)")
plt.title("Inference␣Time␣per␣1-Second␣Segment")
plt.ylim(0, max(times_ms) * 1.2)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

speedup = avg_teacher_time / avg_student_time
print(f"Speedup:␣{speedup:.1f}x")


assert len(teacher_times) == len(student_times), "Mismatch␣in␣
   timing␣data␣length"

fig, axs = plt.subplots(2, 1, figsize=(14, 8), sharex=True)

axs[0].plot(teacher_times, color='blue', label='CNN14␣Inference
   ␣Time')
axs[0].set_ylabel('Time␣(seconds)')
axs[0].legend()
axs[0].grid(True)

axs[1].plot(student_times, color='green', label='Student␣Model␣
   Inference␣Time')
axs[1].set_xlabel('Sample␣Index')
axs[1].set_ylabel('Time␣(seconds)')
axs[1].legend()
axs[1].grid(True)

plt.tight_layout()
plt.show()

fig, axs = plt.subplots(2, 1, figsize=(12, 8))

sns.boxplot(data=data, x="PredictedClass", y="TeacherTime", ax=
   axs[0], palette="tab10")
axs[0].set_title("Teacher␣Model␣Inference␣Time␣Distribution␣per
   ␣Class")
axs[0].set_ylabel("Time␣(seconds)")

sns.boxplot(data=data, x="PredictedClass", y="StudentTime", ax=
   axs[1], palette="tab10")
axs[1].set_title("Student␣Model␣Inference␣Time␣Distribution␣per
   ␣Class")
```

```
axs[1].set_ylabel("Time␣(seconds)")

plt.tight_layout()
plt.show()
```

Bibliography

[1] Altwlkany, K., Hadžić, H., Kurić, A., Lacic, E., “Knowledge distillation for real-time classification of early media in voice communications”, in 2024 32nd International Conference on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, 2024, str. 1–4.

[2] Kong, Q., Cao, Y., Iqbal, T., Wang, Y., Wang, W., Plumbley, M. D., “Panns: Large-scale pretrained audio neural networks for audio pattern recognition”, IEEE/ACM Transactions on Audio, Speech, and Language Processing, Vol. 28, 2020, str. 2880–2894.

[3] Gemmeke, J. F., Ellis, D. P., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., Plakal, M., Ritter, M., “Audio set: An ontology and human-labeled dataset for audio events”, in 2017 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, 2017, str. 776–780.

[4] Hasbullah, H., Said, A. M., Nisar, K., “The effect of echo delay on voice quality in voip network”, in Proceedings of the IASTED International Conference, Vol. 1, No. 9, 2009, str. 200.

[5] Amakuha, “Sip session setup example.svg”, https://commons.wikimedia.org/wiki/File:SIP_session_setup_example.svg, 2022.

[6] Loi, T., “Sip 180 vs 183 vs early media”, https://thanhloi2603.wordpress.com/2017/08/05/sip-180-vs-183-vs-early-media/, 2017.

[7] Gou, J., Yu, B., Maybank, S. J., Tao, D., “Knowledge distillation: A survey”, International Journal of Computer Vision, Vol. 129, No. 6, 2021, str. 1789–1819.

[8] Mljar - AutoML for Humans, “Random forest bagging illustration”, https://commons.wikimedia.org/wiki/File:Random_Forest_Bagging_Illustration.png, 2019.

[9] Son, P. H., Jha, S., Kumar, R., Chatterjee, J. M. *et al.*, “Governing mobile virtual network operators in developing countries”, Utilities Policy, Vol. 56, 2019, str. 169–180.

[10] Camarillo, G., Schulzrinne, H., “Early media and ringing tone generation in the session initiation protocol (sip)”, Tech. Rep., 2004.

[11] Goode, B., “Voice over internet protocol (voip)”, Proceedings of the IEEE, Vol. 90, No. 9, 2002, str. 1495–1517.

[12] Schulzrinne, H., “Sip: Session initiation protocol”, IETF RFC 3261, 2002.

[13] Saint-Andre, P., “Rfc 9280: Rfc editor model (version 3)”, 2022.

[14] Bugatti, A., Flammini, A., Migliorati, P., “Audio classification in speech and music: a comparison between a statistical and a neural approach”, EURASIP Journal on Advances in Signal Processing, Vol. 2002, 2002, str. 1–7.

[15] Lavner, Y., Ruinskiy, D., “A decision-tree-based algorithm for speech/music classification and segmentation”, EURASIP Journal on Audio, Speech, and Music Processing, Vol. 2009, 2009, str. 1–14.

[16] Law, E., West, K., Mandel, M. I., Bay, M., Downie, J. S., “Evaluation of algorithms using games: The case of music tagging.”, in ISMIR, 2009, str. 387–392.

[17] Bertin-Mahieux, T., Ellis, D. P., Whitman, B., Lamere, P., “The million song dataset.”, in Ismir, Vol. 2, No. 9, 2011, str. 10.

[18] Choi, K., Fazekas, G., Sandler, M., “Automatic tagging using deep convolutional neural networks”, arXiv preprint arXiv:1606.00298, 2016.

[19] Hershey, S., Chaudhuri, S., Ellis, D. P., Gemmeke, J. F., Jansen, A., Moore, R. C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B. *et al.*, “Cnn architectures for large-scale audio classification”, in 2017 ieee international conference on acoustics, speech and signal processing (icassp). IEEE, 2017, str. 131–135.

[20] Yu, H., Chen, C., Du, X., Li, Y., Rashwan, A., Hou, L., Jin, P., Yang, F., Liu, F., Kim, J. *et al.*, “Tensorflow model garden”, GitHub, 2020.

[21] Mohammed, K. K., Abd El-Latif, E. I., El-Sayad, N. E., Darwish, A., Hassanien, A. E., “Radio frequency fingerprint-based drone identification and classification using mel spectrograms and pre-trained yamnet neural”, Internet of Things, Vol. 23, 2023, str. 100879.

[22] Hinton, G., Vinyals, O., Dean, J., “Distilling the knowledge in a neural network”, arXiv preprint arXiv:1503.02531, 2015.

[23] Anisimov, N., Olkhovsky, M., Kishinsky, K., “Answering machines detection in contact centers using convolutional neural networks”, URL: http://www. fiztech-usa. net/anisimov/papers/answering-machinesdetection_6. pdf.

[24] Altwlkany, K., Delalić, S., Selmanović, E., Alihodžić, A., Lovrić, I., “A recurrent neural network approach to the answering machine detection problem”, in 2024 47th MIPRO ICT and Electronics Convention (MIPRO). IEEE, 2024, str. 85–90.

[25] Xia, X., Togneri, R., Sohel, F., Huang, D., “Random forest classification based acoustic event detection”, in 2017 IEEE International Conference on Multimedia and Expo (ICME). IEEE, 2017, str. 163–168.

[26] Arora, R., Jain, R., “Voice over ip: Protocols and standards”, Network Magazine, Vol. 23, 1999.

[27] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E., “Sip: session initiation protocol”, Tech. Rep., 2002.

[28] Ho, T. K., “Random decision forests”, in Proceedings of 3rd international conference on document analysis and recognition, Vol. 1. IEEE, 1995, str. 278–282.

[29] Friedman, J., “The elements of statistical learning: Data mining, inference, and prediction”, (No Title), 2009.

[30] Probst, P., Wright, M. N., Boulesteix, A.-L., “Hyperparameters and tuning strategies for random forest”, Wiley Interdisciplinary Reviews: data mining and knowledge discovery, Vol. 9, No. 3, 2019, str. e1301.