VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTEGRACE AUGMENTACE DAT DO PYTORCH

DATA AUGMENTATION INTEGRATION INTO PYTORCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LADISLAV VAŠINA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. IGOR SZŐKE, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

150975

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Vašina Ladislav**
Program:	Informační technologie
Název:	**Integrace augmentace dat do Pytorch**
Kategorie:	Zpracování signálů
Akademický rok:	2023/24

Zadání:

1. Nastudujte Pytorch toolkit - zkuste si základní tutoriály pro trénování modelu (např. MNIST). Zaměřte se na dataloader třídy.
2. Nastudujte existující nástroje pro augmentaci zvukových dat.
3. Navrhněte a implementujte vhodný způsob integrace augmentace do Pytorch.
4. Experimentujte s augmentací dat na zvoleném systému pro automatický přepis řeči.
5. Zhodnoťte dosažené výsledky a navrhněte směry dalšího vývoje.
6. Vytvořte A2 plakátek nebo 20-30 vteřinové video o Vašem projektu.

Literatura:

- Daniel S. Park et al. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition", Interspeech 2019, https://arxiv.org/abs/1904.08779
- Tom Ko et al. "Audio augmentation for speech recognition", Interspeech 2015
- Tom Ko et al. "A Study on Data Augmentation of Reverberant Speech for Robust Speech Recognition", ICASSP 2017
- Nástroj https://github.com/iver56/audiomentations
- Podle pokynů školitele

Při obhajobě semestrální části projektu je požadováno:
Body 1, 2, a část bodů 3, 4 ze zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	9.5.2024
Datum schválení:	9.11.2023

Abstrakt

Tato práce představuje nástroj, který tvoří sjednocené, jednoduché a uživatelsky přívětivé rozhraní nad knihovnami pro augmentaci zvukových dat, jež je možné využít spolu s knihovnou PyTorch. Implementovaný nástroj nabízí možnost použití širokého spektra augmentací z různých knihoven a umožňuje je jednoduše aplikovat na datové sady. Podpory takto velkého spektra augmentací by bylo možné dosáhnout pouze za použití mnoha rozhraní jednotlivých knihoven. Nástroj je schopný od uživatele přijímat seznam augmentací s jejich parametry a sám rozhoduje, jakou z integrovaných knihoven pro dané augmentace použít. Vytvořený nástroj byl testován na úkolu ladění automatického rozpoznávače řeči Whisper. Hlavním přínosem této práce je implementace řešení velkého množství knihoven pro augmentaci zvukových dat, kde každá knihovna poskytuje jiný počet a různé druhy augmentací zvuku a zároveň má i jiné vlastnosti a rozhraní.

Abstract

This thesis presents a tool that creates a unified, simple, and user-friendly interface on top of the audio augmentation libraries that can be used in conjunction with PyTorch library. The implemented tool offers the possibility to use a wide spectrum of augmentations from different libraries and offers easy application of those augmentations on the datasets. The support of the large range of augmentations could be only achieved by using multiple interfaces of the individual libraries. The tool can receive a list of augmentations from the user with its parameters and then it decides which of the integrated libraries it should use to apply that specific augmentation. The created tool was tested on the task of fine-tuning the automatic speech recognition system called Whisper. The main contribution of this work is that it provides a solution to a large number of libraries for the augmentation of audio data, where each library provides a different number and types of augmentations of audio, while also having different features and interfaces.

Klíčová slova

augmentace zvukových dat, integrace augmentačních nástrojů, impulsní odezva místnosti, PyTorch, automatické rozpoznání řeči, OpenAI Whisper

Keywords

audio data augmentation, augmentation tools integration, room impulse response, PyTorch, automatic speech recognition, OpenAI Whisper

Citace

VAŠINA, Ladislav. *Integrace augmentace dat do Pytorch.* Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Igor Szőke, Ph.D.

Integrace augmentace dat do Pytorch

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Igora Szőkeho Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ladislav Vašina
4. května 2024

Poděkování

Rád bych poděkoval svému vedoucímu práce panu Ing. Igorovi Szőkemu Ph.D. za všechny rady, konzultace a připomínky k vyvíjenému nástroji a textu práce.

Obsah

Kapitola 1

Úvod

Modely umělé inteligence, umožňující automatický přepis mluvené řeči do textu, vyžadují k jejich vytvoření použití velkého množství trénovacích dat. Pro vytvoření robustních modelů umožňujících kvalitní přepis mluvené řeči v reálném prostředí, nejen v laboratorním, je potřeba mít rozmanitou trénovací datovou sadu. Získání této rozmanitosti pouhým sběrem dat je ovšem velmi pracné. Z tohoto důvodu je běžným přístupem pro rozšíření trénovacích datových sad generování augmentovaných dat. Tato augmentovaná data obsahují různé nedokonalosti, jež se mohou v mluvené řeči z reálného prostředí nacházet.

Nejrozšířenější knihovnou pro implementaci a trénování modelů umělé inteligence je PyTorch. Tato knihovna jazyka Python bohužel nabízí vývojářům pouze nástroj obsahující minimální množství základních augmentací. Z tohoto důvodu existují komunitou vývojářů vytvořené knihovny, obsahující rozšířenou množinu augmentací, jež se nenachází v interním nástroji knihovny PyTorch. Vetší množství knihoven implementujících různé augmentace ovšem přináší ten problém, že každá knihovna disponuje různými vlastnostmi. Např. knihovnám chybí některé augmentace nebo se liší v podpoře zařízení, na kterých je možné provádět výpočet daných augmentací.

Cílem této práce je vytvořit nástroj, který umožní využívat společné rozhraní nad více knihovnami, jež implementují augmentace zvukových dat. Výsledný nástroj tak poskytne vývojářům prostředek pro komplexní manipulaci s augmentacemi zvukových dat, bez nutnosti pracovat s jednotlivými knihovnami odděleně, při zachování výhod jednotlivých knihoven. To povede k jednoduššímu vývoji robustnějších a kvalitnějších modelů pro přepis mluvené řeči, které lépe obstojí v reálných podmínkách.

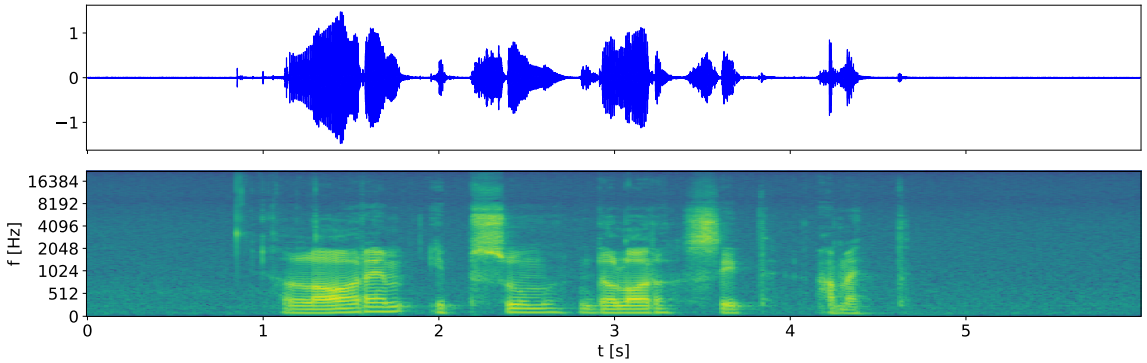
V této práci je nejdříve v kapitole č. 2 popsán důvod využívání augmentací dat v kontextu trénování modelů umělé inteligence a následně jsou předvedeny základní augmentace zvukových dat, které mohou být využity při trénování modelů, jež umožňují přepis mluvené řeči. Následně jsou ve 3. kapitole představeny existující nástroje, pomocí kterých je možné aplikovat různé augmentace na zvukové nahrávky. Jsou zde uvedeny vlastnosti těchto nástrojů, jak z oblasti podporovaných augmentací, tak z oblasti možných zařízení, na kterých lze augmentace provádět. V další kapitole č. 4 je navrženo řešení jednotného rozhraní pro augmentaci dat. Toto řešení se skládá z výběru vhodných nástrojů, které se zkombinují do jednoho rozhraní pro nejrozmanitější výběr augmentací zvuku. Také jsou v této kapitole navrženy komponenty, ze kterých bude finální nástroj sestaven a implementován. Kapitola č. 5 představí různé technologie, které byly využity při implementaci nástroje navrženého v této práci. Dále jsou zde uvedeny i samotné detaily implementace nástroje pro augmentaci zvukových dat. V poslední kapitole č. 6 jsou popsány všechny experimenty, které byly vykonány s využitím vytvořeného nástroje pro augmentaci zvukových dat.

Kapitola 2

Augmentace zvukových dat

Augmentace dat přináší různé techniky, jak uměle navyšovat množství dat v datových sadách, jež se využívají pro strojové učení. Použitím augmentací vnášíme do dat různorodost, která často v datech chybí, protože proces sběru většího množství dat je buďto časově, fyzicky, finančně či z jiných důvodů obtížný. Pokud je pro trénování modelu využita datová sada, jež obsahuje pouze malé množství různorodých vzorků dat, může se stát, že natrénovaný model nebude schopný ve svých předpovědích dostatečně zobecňovat, což znamená, že se naučí každý malý detail z trénovací datové sady a nedokáže tak správně vyhodnotit testovaná data, která se budou více lišit od dat trénovacích [10]. Augmentace tedy umožňuje nejen navyšovat robustnost natrénovaných modelů, ale i předcházet přetrénování, které je způsobeno nedostatečnou rozmanitostí dat v datasetu.

Techniky, jak datové sady rozšiřovat, se primárně liší dle druhu dat. Pokud se snažíme rozšiřovat datové sady obsahující zvuková data, tak se v praxi využívají augmentace samotného zvukového signálu 2.1 a augmentace spektrogramu zvukového signálu 2.2. V dalších dvou podkapitolách budou představeny jednotlivé způsoby augmentací dat. Pro vizuální představení augmentací zvuku bude využita a upravována nahrávka obsahující namluvenou frázi „*Lorem ipsum dolor sit amet*“. Na obrázku 2.1 můžeme vidět dva způsoby vizualizace zvuku, a to zobrazením časového průběhu zvukových vln nahrávky nebo zobrazením Mel [14] spektrogramu zvukového signálu.



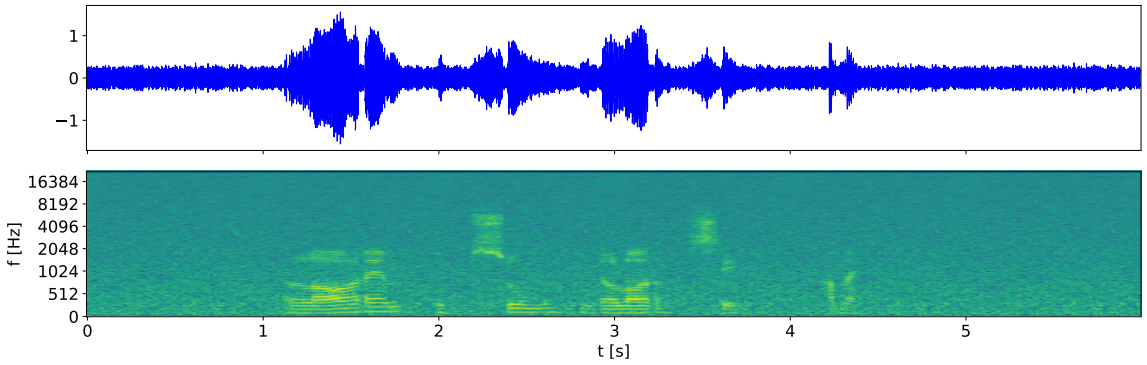
Obrázek 2.1: Časový průběh a spektrogram původní nahrávky. Zvuková vlna je graficky znázorněna jako sinusoida měnící se v čase. Vrchol vlny představuje největší tlak vzduchu a spodní část vlny zobrazuje nejmenší tlak vzduchu. Spektrogram zobrazuje intenzitu frekvenčních složek zvukového signálu v čase. Na ose X je zobrazena frekvence zvuku a osa Y zobrazuje čas.

2.1 Augmentace zvukového signálu

Prvním přístupem je augmentace samotného zvukového signálu. Do této kategorie patří např. přidávání šumů nebo hluků do pozadí nahrávky, inverze polarity signálu, modulace hlasitosti, reverberace, změna rychlosti nahrávky, přidání dalšího řečníka, posunutí výšek apod.

Přidání šumu ve formě obecného signálu

Přidání šumu ve formě obecného signálu do zvukové nahrávky může pomoci k větší přesnosti modelu při jeho setkání s reálnými situacemi, ve kterých nemusí být zvuk vždy čistý a může být ovlivněn různými druhy šumu z okolí [5]. Techniky přidávání šumu jsou různé. Jednou z nejjednodušších technik je přidání náhodného šumu (také známý jako bílý šum) do nahrávky. Pod tímto termínem je myšlen náhodný signál, jež má stejnou energii napříč všemi frekvenčními pásmy. Intenzitu šumu lze upravovat tak, aby bylo dosaženo požadovaného poměru signálu k šumu. Pokud bude intenzita šumu nastavena moc nízko, nebude mít na trénování skoro žádný efekt. Pokud se ale intenzita šumu nastaví příliš vysoko, tak se z takto modifikovaných dat nebude model schopný učit [16].



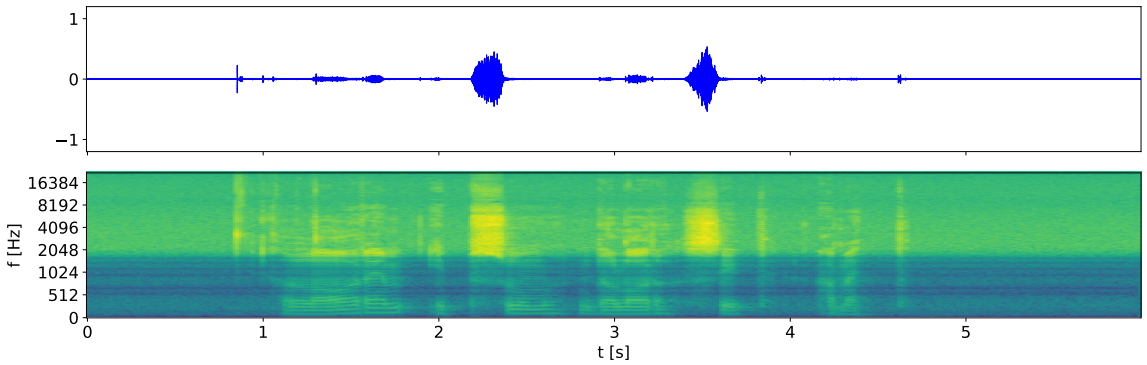
Obrázek 2.2: Zvuková nahrávka s přidaným bílým šumem.

Frekvenční filtry

Frekvenční filtry umožňují propouštět pouze určité frekvence signálu a blokovat ostatní. Těchto filtrů existuje několik a liší se právě v tom, jaké frekvence propouští a které blokují. Mezi nejběžnější frekvenční filtry patří:

- Dolní propust – propouští pouze nízké frekvence, vyšší blokuje.
- Horní propust – propouští pouze vysoké frekvence, nižší blokuje.
- Pásmová propust – propouští pouze frekvence v určitém pásmu. Toto pásmo je určené dvěma mezními frekvencemi.
- Pásmová zádrž – propouští všechny frekvence kromě určitého pásma frekvencí.

Např. využití horní propusti může být výhodné při simulovaní použití vysílaček pro přenos řeči.



Obrázek 2.3: Zvuková nahrávka s aplikovanou horní propustí. Na spektrogramu můžeme vidět, že vyšší frekvence mají v takto augmentované nahrávce vyšší energii.

Konvoluce s impulsní odezvou místnosti

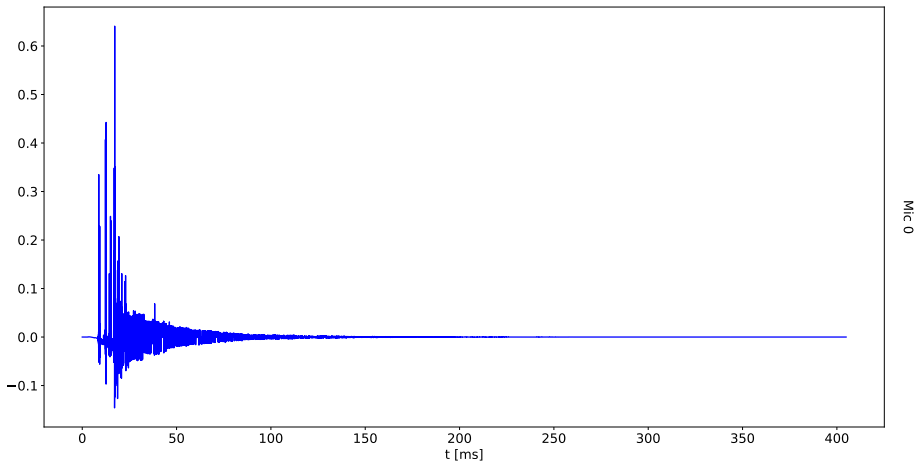
Často se může stát, že v datové sadě využívané pro trénování modelu na rozpoznávání řeči máme pouze nahrávky zaznamenané tak, že obsahují pouze minimální ozvěnu místnosti, ve které byly namluveny. Tento fakt může snižovat přesnost systémů pro automatický přepis řeči, který se setká s nahrávkami, jež obsahují větší množství ozvěn. Způsob, jak tento nežádoucí vliv ozvěny řešit, je využití aplikace konvoluce impulsní odezvy různých místností se zvukovými nahrávkami v datové sadě [15].

Impulsní odezva místnosti je také známá jako Room Impulse Response (RIR). Lze ji měřit několika způsoby a momentálně nejkvalitnějším způsobem měření je využití exponenciálního sinusového posuvu (angl. Exponential Sine Sweep – ESS). ESS je signál, který v sobě obsahuje různé frekvence, které jsou přehrávané v nějakém časovém rozmezí. Tento signál je přehrán v místnosti a postupně je mikrofony zaznamenávána interakce tohoto signálu s prostředím místnosti, včetně odrazů od všech stěn, stropu, podlahy a odrazů od veškerých objektů nacházejících se v konkrétním prostoru. Zaznamenanou odezvu je následně nutné dekonvoluovat ze signálu ESS. Tímto procesem je odstraněn signál ESS z měřené odezvy, čímž získáme izolovanou impulsní odezvu místnosti.

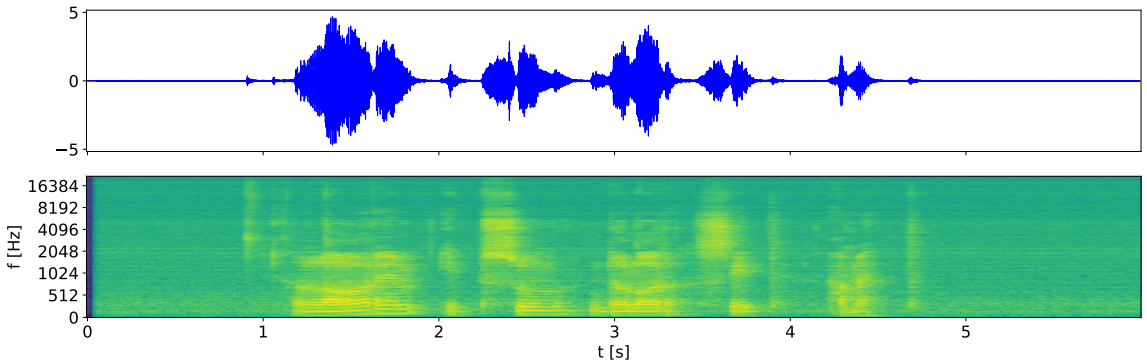
Vztah signálu s aplikovaným dozvukem a původním signálem lze vyjádřit následující rovnicí:

$$y(t) = x(t) * h(t) + a(t) \tag{2.1}$$

kde $y(t)$ je výsledný signál s aplikovaným dozvukem, $x(t)$ je původní signál bez dozvuku, $*$ je symbol konvoluce, $h(t)$ je impulsní odezva místnosti a $a(t)$ je např. přidaný bílý šum.



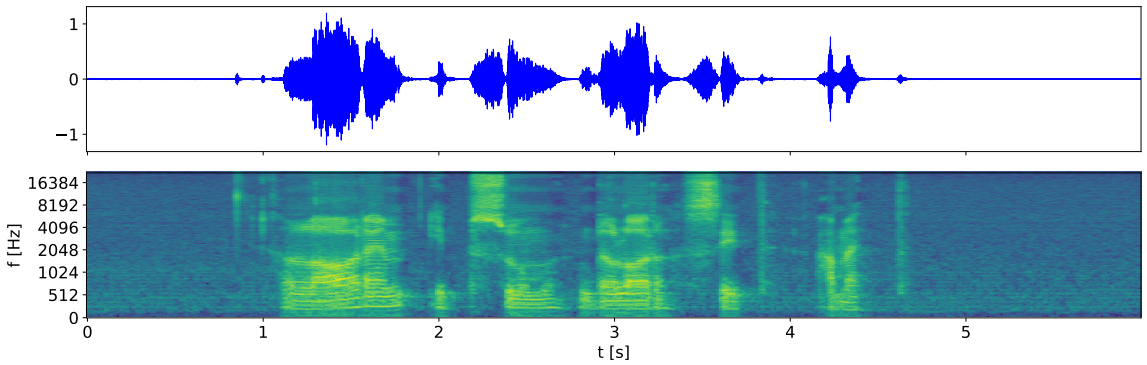
Obrázek 2.4: Signál impulsní odezvy zobrazený v čase.



Obrázek 2.5: Původní zvuková nahrávka konvoluovaná s impulsní odezvou místnosti.

Posunutí výšky základního tónu

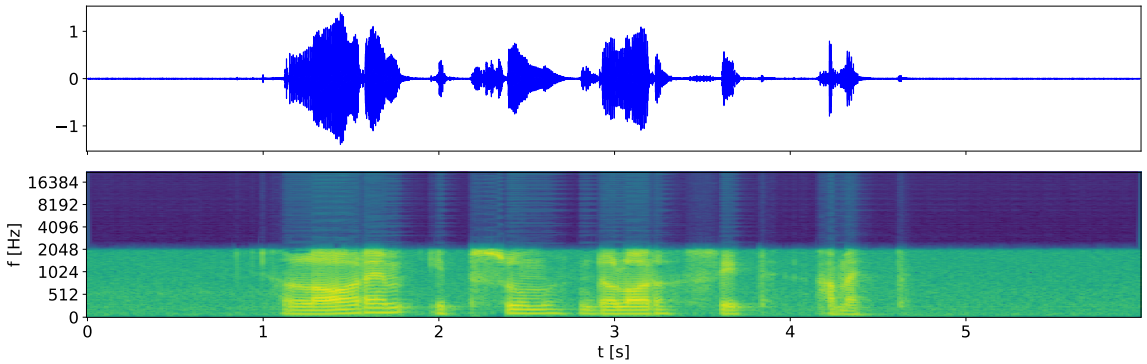
Posunutí výšky základního tónu (angl. pitch shifting) umožňuje měnit frekvenci zvukové nahrávky tak, že každá frekvenční komponenta zvuku je posunuta nahoru či dolů o nějakou konstantu, aniž by byla ovlivněna její rychlost. Použití této augmentace umožňuje modelům se lépe vypořádat s různými hlasy, což umožní modelu lépe fungovat pro širší škálu uživatelů. Nicméně je důležité zvolit vhodnou míru posunutí tónu, aby byl výsledek augmentace stále přirozený a kvalitní. Posunutí výšky tónu v rozsahu ±20 % nebo ±30 % se ukázalo velice efektivní v kontextu rozpoznávání zpěvu [12].



Obrázek 2.6: Zvuková nahrávka s posunutým základním tónem.

Audio kodeky

Audio kodeky jsou softwarové nebo hardwarové komponenty, jež implementují algoritmy pro kompresi a dekompresi zvukových dat. Zvukové kodeky se využívají v kontextu řeči primárně pro snížení bitové rychlosti audio dat. Toto se děje z důvodu šetření šířky pásma potřebného pro přenos dané řeči. Snížením bitové rychlosti přenášené řeči může být ale velmi ovlivněna i její kvalita, což je právě možné využít při augmentaci zvukových dat, při simulaci přenosového kanálu řeči.



Obrázek 2.7: Zvuková nahrávka s aplikovaným kodekem MP3 (8 kbit/s).

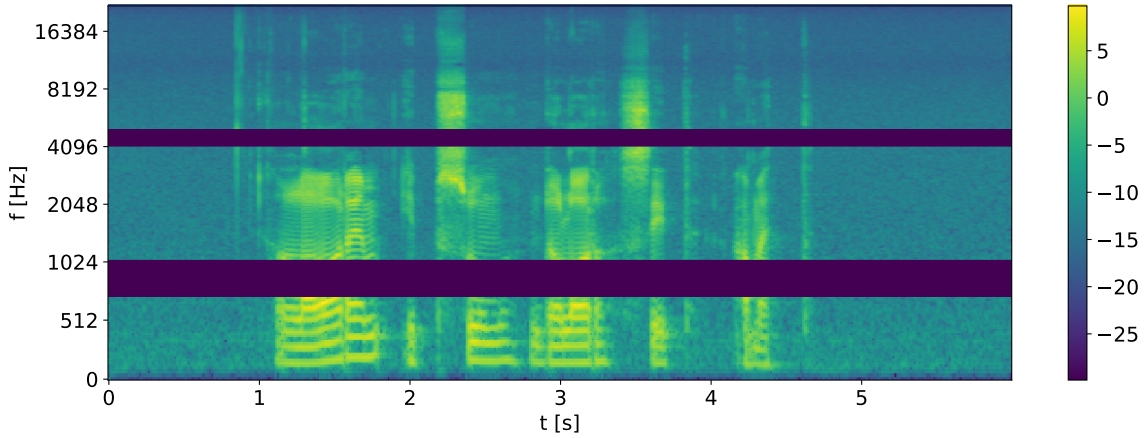
2.2 Augmentace spektrogramu

Trénování neuronových sítí na zvukových datech vyžaduje, aby byly nejdříve všechny nahrávky převedeny do formy spektrogramu, na kterém je možné využít vlastností nejen konvolučních neuronových sítí [6]. Druhým způsobem augmentace zvukových dat je tedy augmentace spektrogramu. Tento inovativní přístup v oblasti augmentace zvukových dat představil v roce 2019 vědecký tým Google Brain ve své práci *SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition* [7].

Tento způsob augmentace dat manipuluje s Mel spektrogramy, je jednoduchý a výpočetně nenáročný na aplikaci. Použití tohoto primitivního přístupu se ukázalo velmi efektivní při trénování neuronových sítí pro automatické rozpoznávání řeči a umožnilo to dosáhnout výsledků porovnatelných s nejmodernějšími přístupy [7].

Maskování frekvenční domény

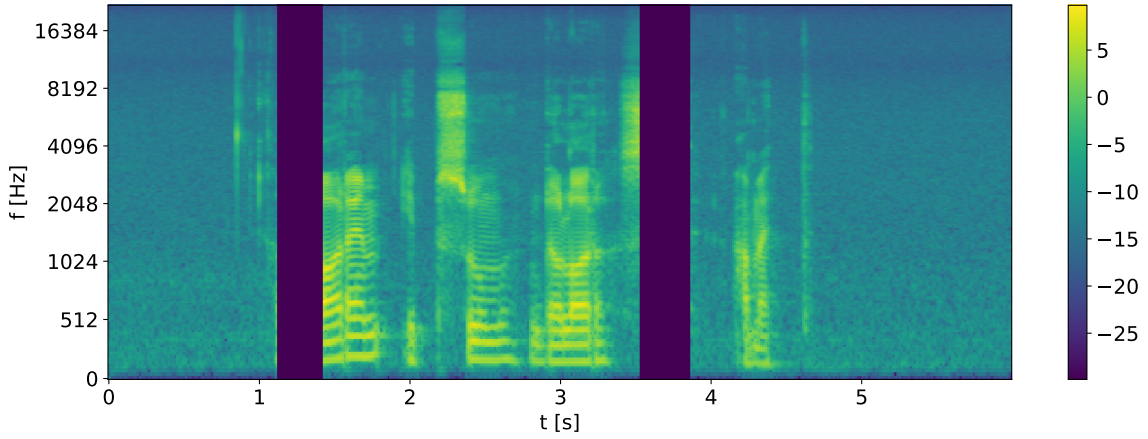
Maskování frekvencí je dosaženo tak, že f po sobě jdoucích Mel frekvenčních kanálů $[f_0, f_0 + f)$ je maskováno. f je vybráno z rovnoměrného rozdělení pravděpodobnosti od 0 po parametr frekvenční masky F a f_0 je vybráno z rozsahu $[0, v - f)$, kde v je počet Mel frekvenčních kanálů [7].



Obrázek 2.8: Mel spektrogram s maskovanou frekvenční doménou.

Maskování časové domény

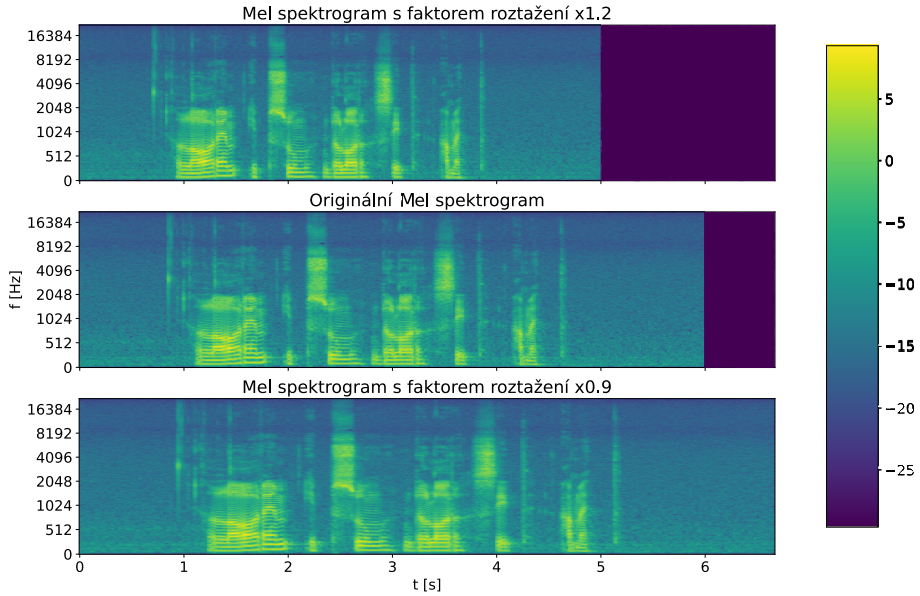
Maskování časové domény je aplikováno tak, že t po sobě jdoucích časových kroků $[t_0, t_0 + t)$ je maskováno. t je vybráno z rovnoměrného rozdělení pravděpodobnosti od 0 po maskovací parametr T a t_0 je vybráno z $[0, \tau - t)$, kde τ je počet časových vzorků [7].



Obrázek 2.9: Mel spektrogram s maskovanou časovou doménou.

Roztažení časové domény

Roztažení časové domény je augmentace, kterou lze aplikovat i na samotný zvukový signál, nejen na Mel spektrogramy. Spočívá v roztažení časové domény dle zvoleného faktoru roztažení bez změny výšky tónu.



Obrázek 2.10: Mel spektrogram s roztaženou časovou doménou.

	Augmentace zvukové nahrávky			**Augmentace spektrogramu**
	Simulace mluvčího	Simulace prostředí	Simulace přenosového kanálu	
Přidání šumu	✗	✓	✓	✗
Aplikace obecného filtru	✗	✗	✓	✗
Konvoluce s RIR	✗	✓	✓	✗
Posunutí výšky základního tónu	✓	✗	✗	✗
Aplikace kodeku	✗	✗	✓	✗
Maskování frekvenční domény	✗	✗	✓	✓
Maskování časové domény	✗	✗	✓	✓
Roztažení časové domény	✗	✗	✗	✓

Tabulka 2.1: Porovnání vybraných augmentací zvuku dle druhu augmentace a dle druhu simulace, kterou může daná augmentace provádět.

Kapitola 3

Existující nástroje pro augmentaci zvukových dat

V současné době je pro augmentaci zvukových dat dostupných několik knihoven jazyka Python, které umožňují různé druhy augmentací. Augmentace mohou být jak primitivní, jako je například přidání šumu do nahrávky, tak i složitější, umožňující například konvoluci vygenerovaných impulzních odezev místností se zvukovými nahrávkami. Dále existují i knihovny, pomocí kterých je možné aplikovat různé audio kodeky. Každá knihovna má jiné vlastnosti, obsahuje různé druhy augmentací a může být i různě optimalizována pro práci s rozhraními pro strojové učení. V následujících podkapitolách budou představeny nejznámější knihovny jazyka Python pro augmentaci zvukových dat.

3.1 librosa

`Librosa`[1] je knihovna, která primárně nabízí robustní rozhraní pro načítaní, ukládání a úpravu zvukových nahrávek. Dokáže převzorkovat nahrávky, měnit měřítka domén spektrogramu či zvukové nahrávky nebo jednoduše zobrazovat zvukové signály a spektrogramy nahrávek. Pro potřebu samotné augmentace zvukových dat tato knihovna nabízí pouze zlomek existujících možností jako např. posunutí výšky tónu 2.1 nebo roztažení časové domény 2.2.

3.2 audiomentations

Knihovnu `audiomentations`[2] je možné použít spolu s knihovnou PyTorch[3] pro augmentaci jak samotné zvukové nahrávky, tak pro augmentaci spektrogramu. Obsahuje široké množství augmentací od jednoduchých, např. přidání šumu 2.1, až po složitější, jako je konvoluce s impulsní odezvou místnosti 2.1. Augmentace je ovšem možné provádět pouze na procesoru. Výhodou knihovny podporující výpočty pouze na procesoru je to, že nevyužívá drahocennou paměť grafických akcelerátorů, kterou je možné využít pro modely s větší velikostí dávek dat. Další výhodou je i možnost provádění transformací, které není možné vykonat na grafických kartách[4], jako např. kompresi zvukové nahrávky do formátu MP3.

[1] https://librosa.org/
[2] https://github.com/iver56/audiomentations
[3] https://pytorch.org/
[4] https://iver56.github.io/audiomentations/guides/cpu_vs_gpu/

3.3 torch-audiomentations

`Torch-audiomentations`[5] je inspirována knihovnou `audiomentations` 3.2, oproti které umožňuje provádět výpočty na grafických akcelerátorech a podporuje přijímání dat ve formě dávek. Obsahuje ale také menší množství augmentací než knihovna `audiomentations`. Hlavní z motivací proč využívat knihovny a nástroje umožňující aplikaci augmentací na grafických akcelerátorech je ta, že při vývoji modelů umělé inteligence nechceme být brzděni příliš pomalou aplikací augmentací trénovacích datových sad. Některé augmentace mohou být totiž výrazně urychleny, pokud je aplikujeme právě s použitím grafického akcelerátoru. Aktuálně je knihovna `torch-audiomentations` ve stálém vývoji a jsou pravidelně přidávány nové augmentace.

3.4 pyroomacoustics

Knihovna `pyroomacoustics`[6] umožňuje vývojáři nasimulovat impulsní odezvy místností (angl. Room Impulse Response – RIR) různých tvarů dle vlastního zadání. Nabízí také širokou volbu materiálů, které lze aplikovat na každou stěnu simulované místnosti, čímž může měnit absorpční koeficienty těchto stěn. Dostupných materiálů v knihovně `pyroomacoustics` je bezmála 100 a materiály se řadí do dvou skupin:

- **Absorpční materiály** – tvoří největší část podporovaných materiálů (cca 90 %) a obsahují materiály jako např. cihlové zdi, dřevěná prkna, koberce, záclony, ale i davy lidí o různých hustotách.
- **Rozptylové materiály** – tvoří zbytek podporovaných materiálů a obsahují např. materiály jako jsou školní lavice nebo difuzní akustické prvky.

Další vlastností této knihovny je schopnost umisťovat více mikrofonů, mikrofonových polí a zdrojů zvuku do libovolného místa ve vygenerované místnosti. Generované impulsní odezvy dokáží následně tato specifická umístění mikrofonů a zdrojů zvuku reflektovat a uživatel má tak možnost si zobrazit impulsní odezvy všech umístěných mikrofonů. Dále tato knihovna dokáže vygenerované impulsní odezvy konvoluovat se zvolenou zvukovou nahrávkou. Knihovna je vybavena intuitivním objektově orientovaným rozhraním jazyka Python a rychlou implementací simulační metody v jazyce C [11].

3.5 pysox

`Pysox`[7] je knihovna, jež obaluje robustní nástroj na úpravu zvuku s názvem SoX[8] (Sound eXchange). Tento nástroj umožňuje aplikovat různé transformace, jako např. převod formátu nahrávek, aplikování efektů nebo dozvuku. Všechny tyto transformace je možné seskupovat a provést nad velkým množstvím dávek dat. SoX se prokázal jako efektivní nástroj při tvorbě různých datových sad. Jeho nevýhodou je lehce nepřehledné a chybám náchylné vytváření příkazů pro vytvoření transformací. Tuto nevýhodu řeší právě knihovna `pysox`, která implementuje jednoduché rozhraní nad nástrojem SoX v jazyce Python [4].

[5] https://github.com/asteroid-team/torch-audiomentations
[6] https://github.com/LCAV/pyroomacoustics
[7] https://github.com/rabitt/pysox
[8] https://sourceforge.net/projects/sox/

3.6 torchaudio

Pro manipulaci se zvukovými daty je možné v rámci knihovny PyTorch využívat její doplňující knihovnu nazvanou `torchaudio`[9]. Tato knihovna obsahuje vestavěné funkce pro načítání a ukládání zvukových souborů. Dále ulehčuje využívání známých datových sad s pomocí jednoduchého rozhraní. Umožňuje také aplikovat některé základní transformace zvukových nahrávek, jako je např. změna hlasitosti, přidání šumu či změna rychlosti. Naopak neumí generování impulsních odezev místností, které by bylo možné využít k vytvoření dozvuku v nahrávce. Knihovna také postrádá implementaci většího množství základních augmentací a neumožňuje si volit rozsahy svých parametrů nebo výběr s jakou pravděpodobností danou augmentaci aplikovat. `Torchaudio` implementuje i novější metody augmentace jako je augmentace spektrogramu zvukové nahrávky 2.2. Další funkcionalitou této knihovny je schopnost využití nástroje SoX, buďto přímo nad zvukovou nahrávkou nebo nad nahrávkou již převedenou do formy tenzoru.

3.7 ffmpeg-python

Knihovna jazyka Python `ffmpeg-python`[10] umožňuje vývojářům programově manipulovat s multimediálními soubory pomocí jednoduchého rozhraní, které je vyvinuto nad velmi rozšířeným nástrojem s názvem FFmpeg[11]. FFmpeg je multiplatformní nástroj, který je otevřeně vyvíjen a umožňuje uživatelům např. kódovat, dekódovat, filtrovat a přehrávat audio či video soubory. V kontextu augmentace řeči pomocí aplikace audio kodeků umožňuje využívat zejména kodeky s nízkým datovým tokem, které simulují výrazné zhoršení přenosového kanálu řeči (např. kodek *AMR*). Díky knihovně `ffmpeg-python` je tedy možné velmi jednoduše využít velké množství funkcí tohoto robustního nástroje a kódovat např. audio nahrávky.

[9] https://pytorch.org/audio
[10] https://github.com/kkroening/ffmpeg-python
[11] https://ffmpeg.org/

Shrnutí vybraných knihoven

V této kapitole byly popsány vybrané knihovny umožňující augmentaci zvukových dat. Některé nástroje se hodí na primitivní úpravy zvuku (`librosa`), jiné zase umožňují provádět složitější augmentace jako je např. simulace prostředí (`pyroomacoustics`) nebo práci na grafických kartách (`torchaudio`). Následující tabulka 3.1 shrnuje jejich hlavní vlastnosti.

	Náhodné parametry augmentací	Velké množství augmentací	Podpora GPU	Generování RIR	SoX	Kodeky
librosa	✗	✗	✗	✗	✗	✗
audiomentations	✓	✓	✗	✗	✗	✗
torch-audiomentations	✓	✗	✓	✗	✗	✗
pyroomacoustics	✗	✗	✗	✓	✗	✗
pysox	✗	✗	✗	✗	✓	✗
torchaudio	✗	✗	✓	✗	✓	✓
ffmpeg-python	✗	✓	✗	✗	✗	✓

Tabulka 3.1: Porovnání klíčových vlastností jednotlivých knihoven.

Kapitola 4

Návrh řešení jednotného rozhraní pro augmentaci zvukových dat

V této kapitole je popsán návrh nástroje, který umožní použití více knihoven pro augmentaci zvukových dat s využitím jednotného rozhraní v jazyce Python. Pomocí tohoto rozhraní mohou vývojáři efektivněji pracovat s různými augmentacemi zvukových dat.

Po zvážení všech výhod a nevýhod jednotlivých knihoven uvedených v kapitole 3 jsem se rozhodl využít augmentace z následujících 5 knihoven:

- **torchaudio** 3.6 – je vybrána, protože již disponuje základními augmentacemi nad spektrogramy zvukových nahrávek a umožňuje provádět augmentace na grafických akcelerátorech. Dále poskytuje funkce pro aplikaci efektů pomocí nástroje SoX[1] a podporuje i aplikaci některých kodeků.

PitchShift	ac3	flac	pcm_mulaw
Speed	adpcm_ima_wav	libmp3lame	pcm_s16le
Vol	adpcm_ms	mp2	pcm_s24le
SpecFrequencyMask	adpcm_yamaha	pcm_alaw	pcm_s32le
TimeMask	eac3	pcm_f32le	pcm_u8
wmav1	wmav2		

Tabulka 4.1: Výpis augmentací využitých z knihovny `torchaudio`.

- **torch-audiomentations** 3.3 – je vybrána z důvodu možnosti provádění většího počtu různých augmentací na grafických akcelerátorech oproti nabídce knihovny `torchaudio`.

AddBackgroundNoise	AddColoredNoise	ApplyImpulseResponse
BandPassFilter	BandStopFilter	HighPassFilter
LowPassFilter	PeakNormalization	PolarityInversion
TimeInversion	Shift	

Tabulka 4.2: Výpis augmentací využitých z knihovny `torch-audiomentations`.

[1] https://sourceforge.net/projects/sox/

- **audiomentations** 3.2 – je vybrána, protože obsahuje největší množství augmentací ze všech zkoumaných knihoven.

AddGaussianNoise	ClippingDistortion	LoudnessNormalization	Padding
AddShortNoises	GainTransition	LowShelfFilter	PeakingFilter
AdjustDuration	HighShelfFilter	Mp3Compression	SevenBandParametricEQ
AirAbsorption	Limiter	Normalize	TanhDistortion

Tabulka 4.3: Výpis augmentací využitých z knihovny `audiomentations`.

- **pyroomacoustics** 3.4 – je vybrána z toho důvodu, že jako jediná knihovna jazyka Python komplexně řeší problém generování impulsních odezev místností.
- **ffmpeg-python**[2] – je vybrán, protože nabízí jednoduché a intuitivní rozhraní nad velmi komplexním nástrojem pro manipulaci s multimediálními soubory. V této práci je využit k aplikaci některých kodeků na zvukové nahrávky. Konkrétně je v mé práci díky této knihovně možné aplikovat kodeky *G.726*, *GSM-FR* a *AMR*.

Celý nástroj jsem navrhl jako kombinaci více komponent, kde každá řeší určité problémy. Pořadí, ve kterém jsou augmentace aplikovány si tvoří uživatel sám a to tím způsobem, že využije komponentu, pomocí které zadá jednotlivé augmentace a jejich konkrétní parametry. Tyto parametry daných augmentací budou moci být také zvoleny z náhodného rozsahu, což přispěje k větší rozmanitosti úrovní jednotlivých augmentací. Náhodný rozsah je u většiny augmentací pocházejících z knihoven `audiomentations` a `torch-audiomentations` vyřešen jejími parametry. Tyto parametry požadují od uživatele vstup minimálních a maximálních hodnot daného parametru, které jsou následně využity danou knihovnou pro vytvoření rozmezí, ze kterého je konečná hodnota náhodně vybrána. Nakonec bude možné zvolit i s jakou pravděpodobností se má daná augmentace provést. Náhodné hodnoty parametrů a pravděpodobnosti aplikace daných augmentací jsou generovány dle rovnoměrného rozdělení pravděpodobnosti.

Augmentace pocházející z vestavěné knihovny `torchaudio`[3] neumožňují ve svém základu inicializovat augmentace s náhodnými parametry nebo je spouštět pouze s určitou pravděpodobností. Z tohoto důvodu bude můj nástroj obsahovat část řešící tento problém. Tato část umožní obalit funkce jednotlivých augmentací knihovny `torchaudio` tak, že je bude možné inicializovat s použitím náhodných hodnot z uživatelem určeného rozsahu a spouštět je se zvolenou pravděpodobností. Tyto náhodné hodnoty a pravděpodobnosti aplikace augmentací, jež pocházejí z knihovny `torchaudio`, budou také vybírány dle rovnoměrného rozdělení pravděpodobnosti, jako tomu je u knihoven `audiomentations` a `torch-audiomentations`.

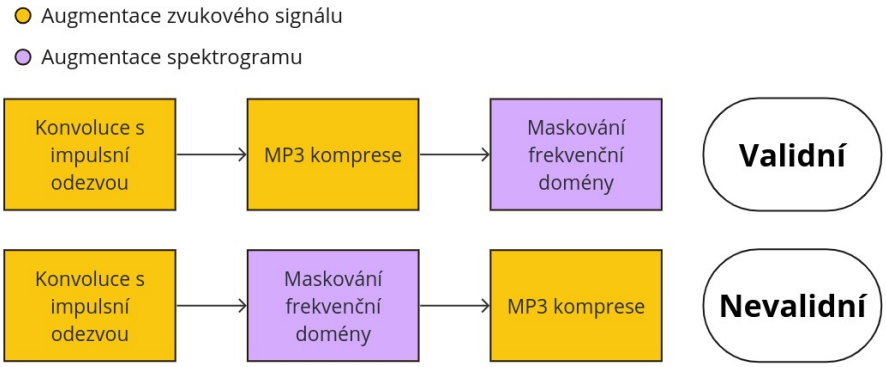
Dalším způsobem jak zvolit augmentace, které mají být aplikovány na datovou sadu, bude možnost zadat pseudo-příkaz či textový soubor s vícero pseudo-příkazy nástroje SoX[4]. Pokud bude uživatelem zvolen soubor jako vstup, zvolí se vybraný pseudo-příkaz náhodně s rovnoměrným rozdělením pravděpodobnosti. Vybraný příkaz bude rozdělen do jednotlivých zvukových efektů s určenými parametry, které jsou v něm obsaženy, pro jednoduší následnou aplikaci.

[2] https://github.com/kkroening/ffmpeg-python
[3] https://pytorch.org/audio
[4] https://sourceforge.net/projects/sox/

Pokud bude chtít uživatel využít augmentaci, pomocí které bude výsledná zvuková nahrávka znít jako by byla pořízena v konkrétním druhu místnosti, využije komponentu, jež tvoří rozhraní nad knihovnou `pyroomacoustics`[5]. Tato komponenta bude přijímat určité parametry, podle kterých bude vygenerována specifická impulsní odezva místnosti. S touto vygenerovanou impulsní odezvou je následně možné, za pomoci knihovny `pyroomacoustics`, konvoluovat vstupní zvukovou nahrávku. Přijímanými parametry mohou být např. souřadnice jednotlivých rohů generované místnosti, výška místnosti, materiály zdí, podlah nebo stropu. Dále bude možné nastavit konkrétní souřadnice zdroje zvukové nahrávky a souřadnice jednotlivých mikrofonů umístěných v místnosti. Dalším přijímaným parametrem bude maximální počet kalkulovaných odrazů zdrojového zvuku. Tímto parametrem bude možné ovlivnit jak rychlost, tak i kvalitu simulace.

Poté, co nástroj přijme a zpracuje uživatelův vstup ve formě zadaných augmentací s určitými parametry, přichází na řadu ta část nástroje, která se již stará o generování finálního seznamu augmentací. Zde bude dle pořadí, ve kterém uživatel augmentace zadal, rozhodnuto o tom, jaké knihovny využít pro aplikovaní dané augmentace.

Pokud uživatel zvolí množinu augmentací, která se má aplikovat vlastním výběrem a nezvolí si možnost aplikace augmentací pomocí pseudo-příkazu SoX, musí brát v úvahu pořadí, ve kterém aplikuje různé druhy augmentací. Augmentace zvukových nahrávek se dělí na dva druhy: augmentace samotného zvukového signálu nahrávky a augmentace spektrogramu zvukové nahrávky. Na základě tohoto dělení augmentací zvuku nesmí uživatel zvolit pořadí augmentací takové, aby se v seznamu augmentací nacházela augmentace samotného zvukového signálu později, než augmentace spektrogramu zvuku.



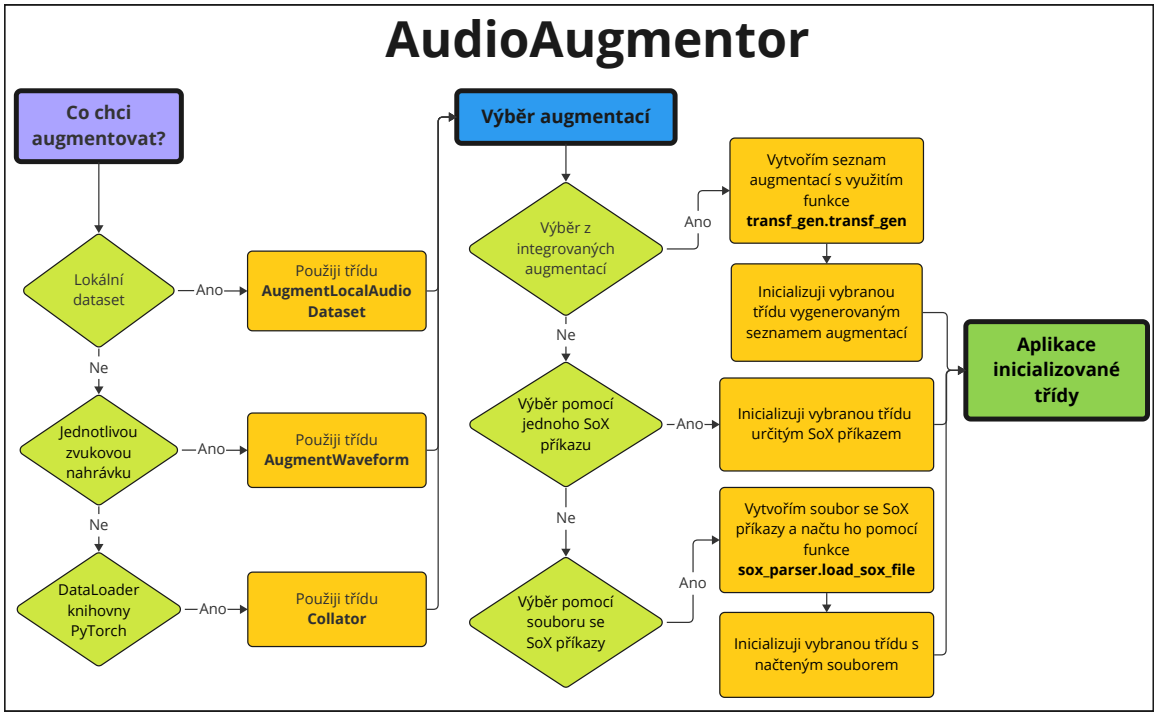
Obrázek 4.1: Ukázka validního a nevalidního pořadí augmentací.

Jádrem mého nástroje je komponenta, která aplikuje augmentace z jednotlivých knihoven dle uživatelského zadání, které je zpracováno výše zmíněnými částmi nástroje. Při inicializaci umožňuje uživateli vybrat preferované zařízení (grafický akcelerátor nebo procesor), na kterém se mají augmentace provádět. Nástroj vyhodnotí každou zvolenou augmentaci a rozhodne, zda je možné ji provádět na zvoleném zařízení či nikoliv. Další funkcionalitou implementované knihovny bude možnost aplikovat augmentace jak na samostatnou zvukovou nahrávku, tak i na uživatelem zvolený adresář obsahující zvukové nahrávky. To umožní jednoduché vytváření lokálních augmentovaných datových sad. Hlavní vlastností nástroje bude možnost aplikovat augmentace i na dávku zvukových dat, což bude možné využít při inicializaci třídy knihovny PyTorch DataLoader 5.1, kterou je možné dále využívat v pro-

[5] https://github.com/LCAV/pyroomacoustics

cesu trénování modelů umělé inteligence. Implementovaný nástroj bude taktéž umožňovat replikaci experimentů s ním provedených. Toto je důležité při výzkumné práci z toho důvodu, aby bylo možné opakovaně dosahovat konzistentních výsledků se stejným nastavením augmentačního nástroje.

Implementovaná knihovna umožňující jednoduchou aplikaci augmentací zvukových dat bude veřejně publikována na platformě PyPi[6] (Python Package Index). PyPi je nejznámější veřejný repozitář balíčků a knihoven pro programovací jazyk Python. Sdílením implementovaného nástroje na tuto platformu se výrazně ulehčí a zrychlí proces jeho stahování, instalace a zprovoznění jednotlivými vývojáři. Vývojáři tak budou moci začít okamžitě využívat jeho funkce pro svoje potřeby a výzkum. Nahraná knihovna bude nést jméno **AudioAugmentor**. Spolu s publikovanou knihovnou budou sdíleny i interaktivní ukázky jak implementovanou knihovnu pro augmentaci zvukových dat správně využívat a to s využitím platformy Google Colab[7] (pro více informací o Google Colab viz kapitolu č. 6).



Obrázek 4.2: Diagram popisující práci s knihovnou AudioAugmentor. Uživatel má možnost výběru druhu zdrojových dat, na které chce aplikovat augmentace. Dle tohoto výběru volí určité třídy, které musí využít. Augmentace je možné zvolit několika způsoby, kde každý způsob využívá jiné funkce knihovny AudioAugmentor.

[6] https://pypi.org/
[7] https://colab.google/

Kapitola 5

Implementace

Samotná implementace navrženého nástroje, který kombinuje knihovny umožňující augmentace zvukových nahrávek do jednotného rozhraní, je hlavním krokem v procesu jeho tvorby. Tato kapitola se zaměřuje na detailní popis realizace jednotlivých modulů a klíčových prvků mého nástroje.

Během implementace byl primárně kladen důraz na uživatelskou přívětivost při využívání finálního nástroje a na jeho modularitu. Jednotlivé moduly byly navrženy tak, aby vývojáři umožnily snadnou integraci nových augmentačních metod do seznamu podporovaných augmentací.

5.1 Použité technologie

K dosažení těchto cílů bylo zapotřebí využívat široké spektrum nástrojů a knihoven jazyka Python. Nástroje popsané v této podkapitole umožnily efektivní vývoj a funkčnost tohoto nástroje pro augmentaci zvukových dat. Pro implementaci mého nástroje byl zvolen programovací jazyk Python[1] ve verzi 3.11.8. Tento programovací jazyk byl zvolen z důvodu jeho popularity a velkého množství knihoven a nástrojů, které umožňují efektivní vývoj, prototypování a testování modelů, jež se zabývají automatickým přepisem mluvené řeči.

Knihovna PyTorch a možnosti načítání dat

Informace o knihovně PyTorch[2], nacházející se v této podkapitole, vycházejí z těchto zdrojů [8, 1, 2]. PyTorch je optimalizovaná knihovna jazyka Python zaměřená na hluboké učení s využitím grafických akcelerátorů a procesorů. Skládá se z následujících komponent:

- **torch** – je knihovna umožňující práci s tenzory podobná knihovně NumPy[3], s výraznou podporou grafických akcelerátorů pro rychlejší výpočty.
- **torch.autograd** – poskytuje třídy a funkce implementující automatickou diferenciaci libovolných skalárních funkcí.
- **torch.jit** – definuje `TorchScript`, do kterého může být model vytvořený v jazyce Python přepsán. Převedený model je následně možné spustit nezávisle na jazyce Python, např. v samostatném programu v jazyce C++.

[1] https://python.org/
[2] https://pytorch.org/
[3] https://numpy.org/

- **torch.nn** – je komponenta poskytující stavební bloky a nástroje pro tvorbu neuronových sítí. Je úzce spojena s komponentou `torch.autograd`.
- **torch.multiprocessing** – je knihovna umožňující sdílení tenzorů napříč procesy. Její výhodou je plná kompatibilita se standardní knihovnou jazyka Python nazvanou `multiprocessing`[4]. Může být výhodná pro načítání dat.
- **torch.utils** – obsahuje třídu `DataLoader` a další užitečné funkce.

V kontextu načítání dat pro hluboké učení je nejvýznamnější komponentou `torch.utils` a primárně její modul `torch.utils.data`. Jádro tohoto nástroje pro načítání dat tvoří třídy `DataLoader` a `Dataset`.

DataLoader

Třída `DataLoader` představuje iterovatelný objekt nad objektem třídy `Dataset`, jež umožňuje účinné a jednoduché načítání dat. Podporuje úpravu pořadí čtení dat z datové sady, automatické sdružení dat do dávek využívaných pro efektivnější trénování modelů, načítání dat s využitím jednoho či více procesů a automatické připínání paměti, které umožňuje rychlejší načítání dat do grafických procesorů s podporou technologie CUDA[5].

Strategie načítání dat z datové sady ve stylu mapy se může řídit třídou `Sampler`, která reprezentuje iterovatelný objekt nad klíči použité datové sady a určuje tak pořadí načítaných klíčů. Využití této třídy je povoleno jen v případě, pokud není při inicializaci třídy `DataLoader` nastaven parametr `shuffle` na pravdivou hodnotu. Nastavení tohoto parametru na pravdivou hodnotu zamíchá načítaná data na začátku každé trénovací epochy, což pomáhá zabránit tomu, aby se model naučil falešné korelace spojené s pořadím načítaných dat.

Specifikace způsobu, jak vybrané vzorky dat automaticky sdružovat do dávek, se řídí dle funkce `collate_fn`, která může být třídě `DataLoader` předána jako argument při inicializaci. Výchozí funkce `collate_fn`, poskytovaná třídou `DataLoader`, převádí každý prvek pole ve formátu NumPy do formátu *torch.Tensor*.

Načítání dat s využitím více procesů lze aktivovat triviálním nastavením parametru `num_workers` na kladné číslo. Načítání dat s využitím pouze jednoho procesu je výchozí možností a může být preferována v případě, že jsou prostředky pro sdílení dat mezi procesy omezené nebo pokud je možné celou datovou sadu načíst do paměti.

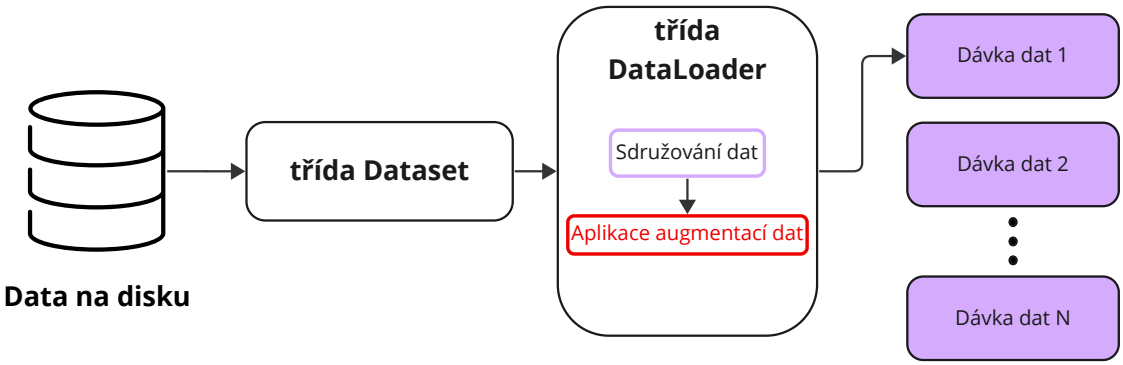
Dataset

Hlavním argumentem konstruktoru výše zmíněné třídy `DataLoader` 5.1 je objekt třídy `Dataset`, který určuje odkud načítat daná data. Knihovna PyTorch podporuje 2 druhy datových sad:

- **Datové sady ve stylu mapy** reprezentuje třída `torch.utils.data.Dataset` a představují mapu klíčů odkazujících na vzorky dat.
- **Iterovatelné datové sady** reprezentuje třída `torch.utils.data.IterableDataset` a takováto forma datové sady je obzvláště užitečná, pokud je náhodné čtení výpočetně drahé a nepravděpodobné a také v případech, kdy velikost dávky závisí na načítaných

[4] https://docs.python.org/3/library/multiprocessing.html
[5] https://developer.nvidia.com/cuda-zone

datech. Takovýto dataset může např. vracet právě čtený proud dat z databáze nebo záznamy generované v reálném čase.



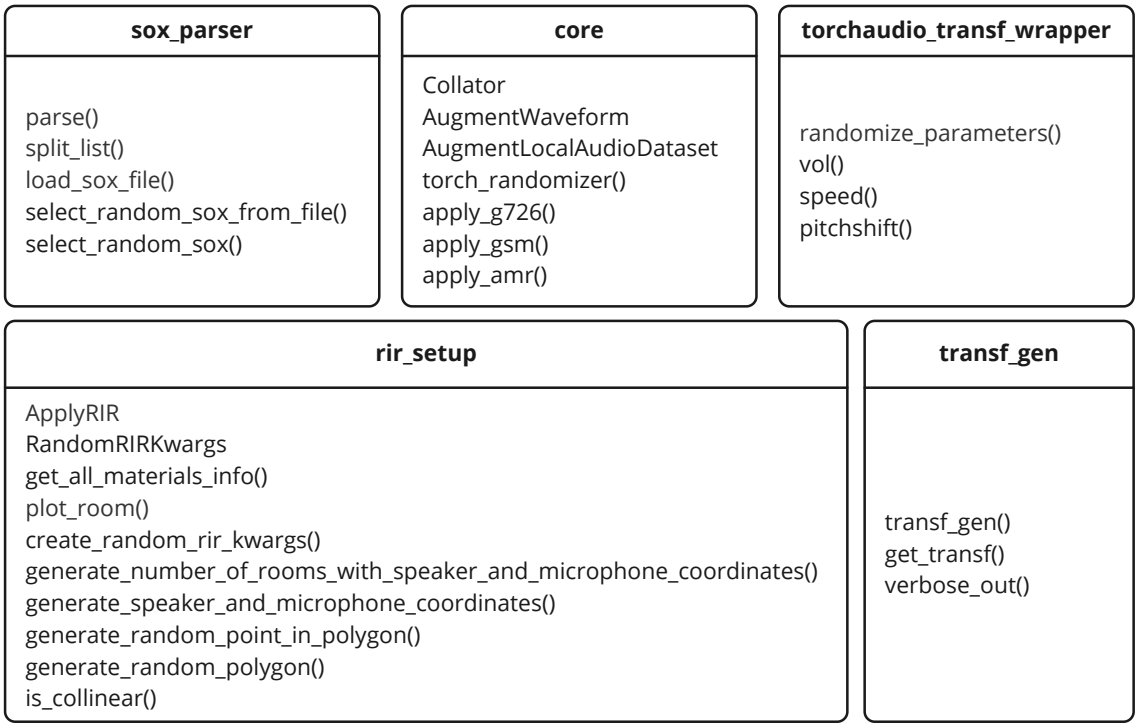
Obrázek 5.1: Schéma zobrazující tok dat v kontextu využití tříd `Dataset` a `DataLoader`.

Knihovna PyTorch umožňuje vývojářům využívat již mnohé zabudované datové sady jako např. datovou sada LibriSpeech[6] nebo CommonVoice[7], které se hojně využívají ve výzkumech k porovnání výkonnosti různých modelů.

[6] https://pytorch.org/audio/stable/generated/torchaudio.datasets.LIBRISPEECH.html
[7] https://pytorch.org/audio/stable/generated/torchaudio.datasets.COMMONVOICE.html

5.2 Implementace modulů nástroje AudioAugmentor

V této podkapitole budou popsány detaily implementace jednotlivých modulů nástroje AudioAugmentor. AudioAugmentor se skládá z několika modulů jazyka Python, kde každý modul zastává jinou funkci v rámci celé implementované knihovny. Následující schéma zobrazuje moduly obsažené v knihovně AudioAugmentor a funkce či třídy, které tyto moduly obsahují.



Obrázek 5.2: Schéma modulů knihovny AudioAugmentor.

Modul transf_gen

S modulem `transf_gen` se uživatel setká při využívání knihovny AudioAugmentor nejspíše jako první. Modul je zodpovědný za generování seznamu augmentací zvuku, podle kterého je následně rozhodnuto jaké augmentace jsou v jakém pořadí aplikovány na uživatelova data. Na počátku tohoto modulu jsou inicializována pole, jež obsahují názvy podporovaných augmentací zvukových nahrávek. Jednotlivé názvy augmentací jsou přiřazeny vždy do pole s názvem, který je odvozený od konkrétní knihovny, ze které bude následně daná augmentace použita. Tyto názvy augmentací, umístěné v polích, zde obsahují odkazy na jejich originální dokumentace, aby si uživatelé knihovny AudioAugmentor mohli popř. vyhledat původní využití jednotlivých augmentací. Nachází se zde i pole, obsahující podporovaný seznam audio kodeků, které je možné pomocí knihovny AudioAugmentor aplikovat.

Modul dále obsahuje funkce `get_tranfs` a `transf_gen`, které poskytují flexibilní a uživatelsky přívětivé rozhraní pro definovaní seznamu požadovaných audio augmentací bez nutnosti rozlišovat, kterou knihovnu (např. torchaudio 3.6 nebo audiomentations 3.2) je potřeba využít k finální aplikaci dané augmentace zvuku.

Uživateli stačí zavolat funkci `transf_gen` jedním z následujících dvou způsobů:

1. Využít názvy podporovaných augmentací jako parametry funkce `transf_gen` a k těmto jednotlivým parametrům přidat slovník jako argument. Klíče tohoto slovníku jsou jednotlivé parametry dané augmentace a hodnoty u těchto klíčů jsou argumenty těchto parametrů dané augmentace.

```
transformations=transf_gen(verbose=True,
                           LowPassFilter={'min_cutoff_freq': 700,
                                          'max_cutoff_freq': 800,
                                          'sample_rate': 16000,
                                          'p': 1},
                           g726={'audio_bitrate': '40k'}
                           )
```

2. Zavolat funkci s využitím řetězce, který obsahuje parametry dané augmentace a argumenty těchto parametrů následují za rovnítkem. Více parametrů dané augmentace v rámci řetězce je mezi sebou odděleno čárkou.

```
transformations=transf_gen(verbose=True,
                           LowPassFilter='''min_cutoff_freq=700,
                                          max_cutoff_freq=800,
                                          sample_rate=16000,
                                          p=1''',
                           g726='audio_bitrate=40k'
                           )
```

Názvy podporovaných augmentací zvuku, které jsou vkládané jako parametry funkce `transf_gen` mohou být napsány jak s velkými, tak s malými písmeny. Nemusí se tedy dodržovat přesný tvar velkých a malých písmen daného názvu augmentace. Pokud chce uživatel aplikovat kodek, který vyžaduje hodnotu parametru *audio_bitrate*, tak je hodnota uživatelova vstupu zkontrolována, zda je zadávána pouze validní hodnota.

Modul `transf_gen` je navržen tak, aby byl použit v kombinaci s dalšími moduly knihovny AudioAugmentor a to primárně s modulem nazvaným `core` 5.2 a jeho třídami `Collator`, `AugmentWaveform` a `AugmentLocalAudioDataset`.

Modul torchaudio_transf_wrapper

Tento modul poskytuje sadu obalových funkcí, které zapouzdřují funkčnost některých augmentací dostupných v knihovně torchaudio 3.6 a umožňují uživatelům aplikovat tyto augmentace s náhodnými parametry, čímž je do procesu augmentace zvukových dat vnášena variabilita a náhodnost. Třídy určené k augmentaci zvukových dat obsažené v knihovně torchaudio totiž ve svém základu neumožňují inicializovat tyto třídy s nějakým náhodným rozsahem hodnot jako to např. umožňují funkce knihovny audiomentations 3.2. Vlastnost těchto obalových funkcí je ta, že kopírují parametry „originálních“ tříd knihovny torchaudio a navíc obsahují volání funkce `randomize_parameters` (také definována uvnitř modulu `torchaudio_transf_wrapper`). Tato funkce přijímá zvolené argumenty obalové funkce jako seznam ve tvaru [1.0, 2.5, 0.1], kde číslo na první pozici znamená dolní hranici pro generování náhodné hodnoty, číslo na druhé pozici znamená horní hranici pro generování náhodné hodnoty a číslo na třetí pozici znamená krok, se kterým se má vygenerovat náhodná hodnota ze zvoleného rozsahu. Pro generování náhodných hodnot je využito rovnoměrné rozdělení

pravděpodobnosti. Po vygenerování náhodných hodnot pro každý zvolený argument jsou takto náhodně vygenerované hodnoty vráceny z této funkce a jsou použity k inicializaci tříd aplikujících augmentace z knihovny torchaudio. Následně jsou tyto inicializované třídy navráceny obalovou funkcí.

Modul core

Modul `core` je hlavní součástí knihovny AudioAugmentor. Obsahuje několik tříd a funkcí, které jsou zodpovědné za aplikaci různých augmentací na zvuková data.
Třídy zodpovědné za aplikaci augmentací jsou 3:

- `Collator`
- `AugmentWaveform`
- `AugmentLocalAudioDataset`

Všechny tyto třídy umožňují již zmiňovanou aplikaci zvolených augmentací na zvuková data, liší se však v tom, jaký druh dat podporují na svém vstupu. Dále jsou navrženy tak, aby dokázaly fungovat s různými knihovnami a nástroji pro zpracovávání zvuku (viz kapitola Návrh 4).

Hlavní třídou tohoto modulu je třída `Collator`. Ta je implementována takovým způsobem, že ji je možné zavolat a použít jako argument třídy `DataLoader` 5.1, což ve výsledku umožňuje integraci knihovny AudioAugmentor spolu s knihovnou PyTorch 5.1. Jelikož je třída navržena tak, aby fungovala jako collate funkce třídy `DataLoader`, přijímá tato třída při svém zavolání data od třídy `DataLoader` ve formě dávek tenzorů.

Před použitím musí být třída nejdříve inicializována následujícím způsobem.

```
collate_fn=core.Collator(
    transformations=transformations, device='cpu', sox_effects=None, sample_rate=16000
)
```

Uvnitř třídy je následně iterováno přes každou zvukovou nahrávku v dávce, která je poskytnuta třídou `DataLoader`. Na základě seznamu augmentací, vygenerovaného pomocí funkce modulu `transf_gen` 5.2, který je poskytnut jako argument parametru *transformation*, jsou postupně aplikovány jednotlivé augmentace obsažené v tomto seznamu. Argumentem *device* si uživatel vybírá zařízení (grafický akcelerátor nebo procesor), na kterém si přeje mít umístěná zvuková data a augmentace, jež budou prováděny. Pokud to právě prováděná augmentace umožňuje, jsou data ponechána na zařízení dle uživatelova výběru. Pokud tomu tak není, jsou přesunuta na zařízení potřebné k aplikaci oné augmentace. Jelikož knihovna AudioAugmentor integruje funkcionality několika různých augmentačních nástrojů a knihoven, vyvstává několik problémů při aplikaci augmentací. Problémem je například to, že se snažíme aplikovat jednotlivé augmentace z různých knihoven a s různými vlastnostmi na jedna vstupní data. Při aplikaci jednotlivých augmentací musí být tedy přesně rozlišováno, jaká je např. zdrojová knihovna či nástroj právě aplikované augmentace, jaké datové typy daná augmentace podporuje na svém vstupu, v jakém tvaru dokáže daná augmentace přijímat data nebo na jakém zařízení se mohou data nacházet, aby bylo možné provést augmentaci. Všechny tyto kontroly a případné změny těchto vlastností vstupních dat musí být provedeny před samotnou aplikací určité augmentace. Kontroly jsou následně provedeny i zpětně, aby se zachovaly stejné vlastnosti zvukových dat jak před, tak po aplikování augmentace.

Aplikace audio kodeků G.726, GSM-FR a AMR probíhá s využitím funkcí `apply_g726`, `apply_gsm` a `apply_amr`, které všechny využívají knihovnu ffmpeg-python[8] a přijímají jako argumenty cesty dočasných souborů pro vstup a výstup. Na místo cesty dočasného souboru pro vstup je z paměti uložena právě augmentovaná zvuková nahrávka a na místo cesty výstupního dočasného souboru je po využití nástroje a aplikaci audio kodeku uložena již augmentovaná nahrávka. Z tohoto výstupního dočasného souboru je augmentovaná nahrávka posléze znovu načtena do paměti a je možné s ní nadále pracovat. Po zpětném načtení do paměti jsou jak vstupní, tak výstupní dočasné soubory smazány, aby zbytečně nezabíraly místo na disku. Jelikož výstupem těchto audio kodeků je signál se vzorkovací frekvencí 8 kHz, může nastat situace, že vstupní zvuková nahrávka bude mít jinou vzorkovací frekvenci než 8 kHz. Při této situaci bude signál zpětně navzorkován na frekvenci, kterou signál měl před aplikací kodeku. Dále je pomocí funkce `torch_randomizer` umožněno aplikovat i funkce knihovny torchaudio 3.6 s určitou náhodností.

Jako další v pořadí tento modul obsahuje třídu nazvanou `AugmentWaveform`. Tato třída se liší od třídy `Collator` primárně tím, že ji je možné po inicializaci zavolat s jednou zvukovou nahrávkou, nad kterou vykoná požadovanou augmentaci. Třída tedy pracuje s jedno-dimenzionálním polem *numpy.ndarray*, které představuje vstupní zvukovou nahrávku a vrací augmentovanou nahrávku ve stejném datovém typu a tvaru.
Třídu inicializujeme následovně:

```
augmentations=core.AugmentWaveform(
    transformations=transformations, device='cpu', sox_effects=None, sample_rate=16000
)
```

Vlastnost této třídy se dá využít například při augmentaci dat bez využití knihovny PyTorch.

Poslední třída, která uživateli umožňuje aplikovat nějakým způsobem augmentace na zvuková data je třída `AugmentLocalAudioDataset`. Tato třída obaluje výše zmíněnou třídu `AugmentWaveform` a umožňuje po její inicializaci tuto třídu zavolat s argumenty *input_dir* a *output_dir*, které značí umístnění vstupního adresáře, obsahujícího zvukové nahrávky v podporovaných formátech (wav, flac, ogg, mp3, aac, opus, m4a, wma, ac3), na které aplikuje augmentace dle uživatelského zadání a uloží je jako samostatné soubory do zvoleného výstupního adresáře.
Třída se inicializuje následujícím způsobem:

```
augmentations=core.AugmentLocalAudioDataset(
    transformations=transformations, device='cpu', sox_effects=None, sample_rate=16000
)
```

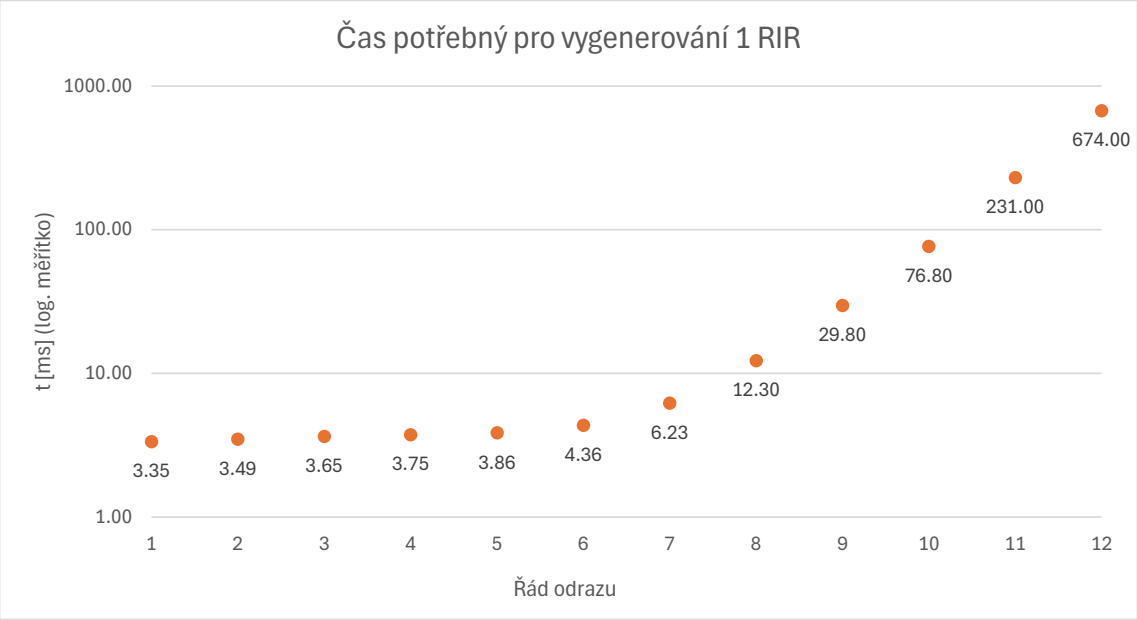
Modul rir_setup

Modul `rir_setup` je primárně zaměřen na využívání funkcí a vlastností knihovny pyroomacoustics 3.4. Je zodpovědný za generování impulsních odezev různých místností dle uživatelského zadání a jejich následnou konvoluci s vytvořeným zdrojem zvuku uvnitř místnosti. Modul tedy umožňuje vytváření a konfiguraci prostorových prostředí, jež simulují různé akustické podmínky, jako jsou např. místnosti s odlišnými materiály stěn a výškami stropů. Tímto způsobem je možné trénovat modely umělé inteligence na širokou škálu akustických podmínek, což ve výsledku zlepší schopnost modelů zobecňovat.

[8] https://github.com/kkroening/ffmpeg-python

Hlavní funkcionalita tohoto modulu je obsažena uvnitř třídy s názvem `ApplyRIR`. Tuto třídu je po inicializaci možné zavolat a jako její parametr se poskytuje zvuková nahrávka. Po jejím zavolání je nejdříve zkontrolován datový typ zvukové nahrávky, která je zadána jako argument a je uložen i její datový typ na počátku volání. Zvuková nahrávka je také navzorkována na vzorkovací frekvenci 16 kHz, což je frekvence, na kterou je pyroomacoustics interně nastaven, aby s ní pracoval. Následně je zkontrolováno, zda je datový typ zvukové nahrávky *torch.Tensor*. Pokud ano, je ještě zajištěna případná změna umístění tenzoru zvukové nahrávky z grafického akcelerátoru na procesor a nakonec je tenzor převeden na datový typ *numpy.ndarray*. V dalším kroku je vytvořen objekt místnosti dle uživatelských parametrů, jsou přidány materiály zdí, podlahy a stropu. Dále jsou do vytvořené místnosti na určené souřadnice (souřadnicová soustava je v metrech) vloženy snímací mikrofon a zdroj zvuku, který je nastaven na vstupní zvukovou nahrávku. Nakonec je spuštěna simulace, která s využitím metody ISM (Image-Source Method) [3] vygeneruje impulsní odezvu specifikované místnosti a provede konvoluci vstupní zvukové nahrávky s touto vygenerovanou impulsní odezvou. Volání třídy `ApplyRIR` vrací vstupní zvukovou nahrávku modifikovanou tak, jako by byla přehrána v požadované místnosti a je také převedena na stejný datový typ a popřípadě zařízení, na němž byla nahrávka umístěna na počátku volání této třídy. Impulsní odezva místnosti je aplikována na celou zvukovou nahrávku. Vrácená nahrávka je také zpětně navzorkována na vzorkovací frekvenci, kterou měla nahrávka na počátku volání třídy `ApplyRIR`. Čas potřebný pro vygenerování jedné impulsní odezvy místnosti je nejvíce závislý na parametru *max_order*. Tento parametr se využívá v rámci metody ISM k nalezení všech obrazových zdrojů až do právě zadané hodnoty. Naměřené časy potřebné pro vygenerování 1 RIR jsou zobrazeny v grafu 5.3. Hodnoty byly měřeny pomocí modulu jazyka Python `timeit`[9]. Zařízení využité k měření prezentovaných hodnot mělo procesor Intel i7-11370H 4,8 GHz a 40 GB DDR4 RAM.

[9] https://docs.python.org/3/library/timeit.html



Obrázek 5.3: Graf zobrazující čas potřebný k vygenerování jedné impulsní odezvy místnosti v závislosti na maximálním řádu odrazu. Osa Y je v logaritmickém měřítku pro lepší přehlednost.

Modul dále obsahuje i funkce, které umožňují dle uživatelského zadání vygenerovat seznam dvou-dimenzionálních souřadnic rohů místností spolu se souřadnicemi mikrofonů a mluvčích umístěných ve vygenerovaném polygonu. Funkce `generate_random_polygon` přijímá jako argument počet vrcholů a rozsahy souřadnic osy X a Y (v metrech), ve kterých má být požadovaná místnost vygenerována. Samotné generování bodů probíhá tak, že se nejdříve v cyklu vygeneruje úhel v radiánech a poloměr, který je určený uživatelem definovaným rozsahem, v jakém se má bod nacházet. Těmito dvěma hodnotami je vygenerován bod v polárních souřadnicích, jenž je následně převeden do kartézských souřadnic a přičtením vygenerované hodnoty z uživatelova rozsahu je bod posunut do správných mezí. Vygenerovaný bod je následně zaokrouhlen a zkontrolován, zda není kolineární s nějakými již dvěma vygenerovanými body. Pokud je bod kolineární (netvoří roh místnosti), je zahozen a bod se vygeneruje znovu. Zda je bod kolineární či nikoliv, je zjištěno pouhým výpočtem plochy trojúhelníku na základě souřadnic jiných, již vygenerovaných dvou bodů a souřadnic nově generovaného bodu. Jestliže vyjde plocha nulová, je bod kolineární.

Dle popisu výše je třídu `ApplyRIR` možné využít jedním z následujících dvou způsobů:

1. Využít přesnou definici parametrů místnosti.

```
rir_kwargs = {
    'audio_sample_rate': 16000,
                    #[[x1,y1],[x2,y2], ...] clockwise
    'corners_coord': [[0, 0], [0, 3], [5, 3], [5, 1], [3, 1], [3, 0]],
    'walls_mat': 'curtains_cotton_0.5',
    'room_height': 2.0,
    'max_order': 3,
    'floor_mat': 'carpet_cotton',
    'ceiling_mat': 'hard_surface',
    'ray_tracing': True,
    'air_absorption': True,
    'source_coord': [[1.0], [1.0], [0.5]], # [[x], [y], [z]]
    'microphones_coord': [[3.5], [2.0], [0.5]], # [[x], [y], [z]]
}
transformations = transf_gen.transf_gen(ApplyRIR=rir_kwargs)
rir_aug=core.Collator( # core.AugmentWaveform or core.AugmentLocalAudioDataset
    transformations=transformations, device='cpu', sample_rate=16000)
```

Zdrojový kód 5.1: Využití třídy `ApplyRIR` s pevně definovanými parametry místnosti.

V tomto případě uživatel pevně definuje souřadnice jednotlivých rohů místnosti s využitím slovníku *rir_kwargs* a jeho klíče *corners_coord*. Souřadnice jednotlivých bodů jsou zadávány ve směru pohybu hodinových ručiček. Dále musí konkrétně specifikovat souřadnice zdrojového zvuku a snímacího mikrofonu s využitím klíčů *source_coord* a *microphones_coord*. Zde se udávají souřadnice ve formátu [[X],[Y],[Z]]. Takto vytvořený slovník je použit jako argument třídy `ApplyRIR` při definici augmentací s využitím funkce `transf_gen`.

2. Využít pro definici parametrů místnosti rozsahy souřadnic a rozsah počtu vrcholů místnosti. Pomocí těchto parametrů bude místnost generována náhodně.

```
rir_kwargs = {
    'audio_sample_rate': sampling_rate,
    'x_range': (0, 10),
    'y_range': (0, 10),
    'num_vertices_range': (3, 6),
    'mic_height': 1.5,
    'source_height': 1.5,
    'walls_mat': 'curtains_cotton_0.5',
    'room_height': 2.0,
    'max_order': 3,
    'floor_mat': 'carpet_cotton',
    'ceiling_mat': 'hard_surface',
    'ray_tracing': True,
    'air_absorption': True,
}
transformations = transf_gen.transf_gen(ApplyRIR=rir_kwargs)
rand_rir_aug=core.Collator( # core.AugmentWaveform or core.AugmentLocalAudioDataset
    transformations=transformations, device='cpu', sample_rate=16000)
```

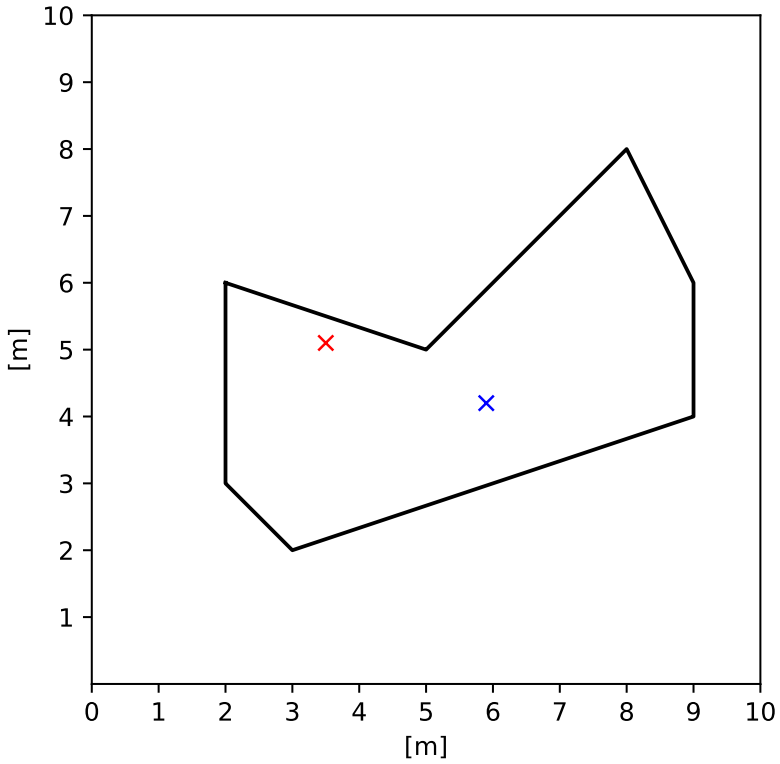
Zdrojový kód 5.2: Využití třídy `ApplyRIR` s nastavenými rozsahy některých parametrů místnosti. Rozsahy jsou použity pro náhodné generování místnosti.

Tento případ se od toho 1. liší tím, že uživatel místo pevných souřadnic rohu místnosti definuje rozsahy osy X a Y (v metrech) pomocí klíčů *x_range* a *y_range*. Dále musí nastavit klíč *num_vertices_range* na rozsah počtu vrcholů, ze kterého bude následně vybrán konkrétní počet vrcholů generované místnosti. Nakonec uživatel definuje pouze výšku zdrojového zvuku a mikrofonu s využitím klíčů *mic_height* a *source_height*.

Dále může uživatel ve slovníku, pomocí kterého inicializuje třídu `ApplyRIR`, uvést např. materiály stropu, zdí či podlahy. Jaké řetězce přiřadit ke klíčům *walls_mat*, *floor_mat* a *ceiling_mat*, zjistí uživatel zavoláním funkce `rir_setup.get_all_materials_info()`. Tato funkce vrací zformátovaný výstup dostupných materiálu v knihovně pyroomacoustics. Detaily dostupných materiálů jsou také dostupné na stránkách dokumentace[10] této knihovny.

```
Massive constructions and hard surfaces:
  hard_surface: {parameters...}
  brickwork: {parameters...}
Floor coverings:
  linoleum_on_concrete: {parameters...}
  carpet_cotton: {parameters...}
```

Zdrojový kód 5.3: Zkrácená a zjednodušená ukázka výstupu funkce poskytující informace o dostupných materiálech.



Obrázek 5.4: Ukázka náhodně vygenerované místnosti pomocí implementované funkce. Křížky uvnitř vygenerovaného tvaru místnosti značí umístění zdroje zvuku a mikrofonu.

[10] https://pyroomacoustics.readthedocs.io/en/pypi-release/pyroomacoustics.materials.database.html

Modul sox_parser

Modul `sox_parser` obsahuje funkce, které umožňují použití jednoho nebo více pseudo-příkazů ve formátu SoX[11] jako argument při inicializaci tříd, které již zajišťují samotnou aplikaci augmentací na uživatelem zadaná data. Příkaz, který chce uživatel využít pro aplikaci augmentací na zvuková data, musí být zadán v jednom z následujících formátů:

1. `--sox="norm gain 0 phaser 0.5 0.6 1 0.45 0.6 -s"`

Formát č. 1 obsahuje pouze samotný pseudo-příkaz formátu SoX obsahující definici jednotlivých efektů, které chce uživatel aplikovat na zvuková data.

2. `--sox="norm gain 20 phaser 0.5 0.6 1 0.45 0.6 -s" g726 audio_bitrate 40k`

Formát č. 2 obsahuje navíc oproti formátu č. 1 i možnost aplikace audio kodeku. Kodek je aplikován až po aplikaci efektů definovaných pseudo-příkazem SoX a není aplikován s využitím příkazů SoX, ale je interně aplikován s využitím buďto knihovny `torchaudio` nebo knihovny `ffmpeg-python`. Specifické argumenty určitých kodeků se uvádějí za názvem kodeku oddělené mezerou.

Konkrétně může uživatel využít 2 způsoby volby augmentací pomocí pseudo-příkazu SoX:

1. Využít samotný pseudo-příkaz, který použije při inicializaci tříd určených pro augmentaci dat

```
sox=core.Collator( # core.AugmentWaveform or core.AugmentLocalAudioDataset
    sox_effects='--sox="norm gain 0 phaser 0.5 0.6 1 0.45 0.6 -s"',
    device='cpu', sample_rate=16000)
```

Pokud uživatel zvolí možnost č. 1, tak budou na všechna data aplikovány stejné efekty definované zvoleným příkazem. Poskytnutý příkaz je funkcí `parse` rozdělen na seznam jednotlivých efektů a jejich parametrů. S takto vytvořeným seznamem je ještě zavolána funkce `split_list`, která získá podporované efekty pomocí funkce `torchaudio.sox_effects.effect_names()`. Na základě podporovaných efektů je vstupní seznam rozdělen na více seznamů, které již obsahují pouze název určitého efektu a jeho argumenty. Pokud příkaz obsahuje navíc i část s definovaným kodekem, je i tato část příkazu rozdělena do dalšího pole.

2. Vytvořit samostatný textový soubor, kde se na každém řádku nachází pseudo-příkaz SoX v jednom z výše uvedených formátů.

```
--sox="norm gain 10 highpass 1500 phaser 0.5 0.6 1 0.45 0.6 -s"
#--sox="norm gain 20 highpass 300 phaser 0.5 0.6 1 0.4 0.6 -s"
--sox="norm gain 20 highpass 300 phaser 0.5 0.6 1 0.45 0.6 -s" pcm_mulaw
--sox="norm gain 20 highpass 300 phaser 0.5 0.6 1 0.45 0.6 -s" amr audio_bitrate 4.75k
```

Zdrojový kód 5.4: Textový soubor obsahující několik pseudo-příkazů ve formátu SoX.

[11] https://sourceforge.net/projects/sox/

```
sox_file_content = sox_parser.load_sox_file('sox_file_example.txt')
sox=core.Collator( # core.AugmentWaveform or core.AugmentLocalAudioDataset
    sox_effects=sox_file_content,
    device='cpu', sample_rate=16000)
```

Zdrojový kód 5.5: Ukázka inicializace augmentační třídy s využitím souboru s pseudo-příkazy ve formátu SoX.

Jestliže uživatel zvolí možnost č. 2, tak musí vytvořený textový soubor s příkazy načíst pomocí funkce `load_sox_file`, které jako argument poskytne cestu k vytvořenému souboru. Tato funkce při načítání obsahu souboru navíc ignoruje zakomentované řádky začínající znakem #. Ze všech následně zpracovaných (nezakomentovaných) řádků obsahujících pseudo-příkazy ve formátu SoX je následně náhodně vybrán pseudo-příkaz, jenž se využije k definici efektů, které mají být aplikovány na zvukovou nahrávku. Náhodný výběr pseudo-efektu je proveden pomocí rovnoměrného rozdělení pravděpodobnosti.

Publikace knihovny AudioAugmentor

Implementovanou knihovnu, umožňující jednoduchou augmentaci zvukových dat, kterou jsem pojmenoval AudioAugmentor, jsem se rozhodl veřejně sdílet na platformě PyPi[12]. Tímto krokem se zvýší její viditelnost a výrazně se ulehčí možnost použití knihovny AudioAugmentor jinými vývojáři či výzkumníky, kteří mají zájem aplikovat augmentace na zvuková data.

Pro úspěšnou publikaci knihovny na platformu PyPi bylo třeba provést několik kroků. Nejdříve jsem si musel založit účet na této platformě. Následně jsem musel v repozitáři, obsahujícím zdrojové kódy, vytvořit soubor *pyproject.toml*, který slouží jako konfigurační soubor pro nástroje jako je např. Twine[13], jenž umožňuje jednoduché publikování na platformu PyPi. Tento soubor obsahuje řadu informací potřebných k publikaci a zabalení zdrojových souborů. Mezi tyto informace patří např. název projektu, odkaz na zdrojové kódy, klíčová slova, se kterými se má knihovna publikovat, seznam závislostí nebo licenci, pod kterou je knihovna publikována.

K nahrání balíčku vytvořeného ze zdrojových kódů jsem využil již zmíněný nástroj Twine. Ten umožňuje vykonat tuto akci pouze pomocí jednoho příkazu.

AudioAugmentor je publikován na adrese https://pypi.org/project/AudioAugmentor/. Nachází se zde popis samotné knihovny, tabulka podporovaných augmentací značící, ze kterých knihoven či nástrojů jsou jednotlivé augmentace využívány. Je zde popsán i proces instalace knihovny nebo ukázky použití knihovny spolu s ukázkou definic všech podporovaných augmentací. Tyto ukázky použití jednotlivých augmentací obsahují i odkazy na jejich dokumentace či definice. Nachází se zde také odkaz na platformu Google Colab 6, kde je připraven sešit typu Jupyter, který obsahuje kompletní instalaci a ukázkové využití knihovny AudioAugmentor, což uživateli významně ulehčí práci a čas při prvotním seznamování se s knihovnou, jelikož uživatel může tento publikovaný sešit okamžitě spustit a může začít prozkoumávat funkcionalitu knihovny AudioAugmentor.

[12] https://pypi.org/
[13] https://pypi.org/project/twine/

Kapitola 6

Experimenty

Tato kapitola obsahuje popis provádění experimentů, které ověřují funkcionalitu implementované knihovny pro augmentaci zvukových dat AudioAugmentor[1]. Nejdříve je popsán systém pro automatické rozpoznávání řeči s názvem Whisper 6, se kterým byly primárně prováděny experimenty k ověření funkčnosti implementované knihovny. Jsou zde také popsány další technologie, jež byly využity pro testování. Následně je předvedeno, že s využitím mé knihovny je opravdu možné zlepšit metriku WER (Word Error Rate) automatického rozpoznávače Whisper. Dále je ukázáno, že je s knihovnou možné replikovat experimenty. Také je porovnána složitost zadávání určité množiny augmentací s využitím samotné knihovny AudioAugmentor oproti využití většího množství jiných knihoven či nástrojů. Nakonec je předvedeno využití implementované knihovny spolu s PyTorch a je předvedena i aplikace vlastních impulsních odezev a šumů.

OpenAI Whisper

Informace uvedené o automatickém rozpoznávači řeči Whisper pocházejí z publikace *Robust Speech Recognition via Large-Scale Weak Supervision* [9], ve které byl Whisper představen výzkumníky z firmy OpenAI[2]. Název Whisper vychází ze zkratky WSPSR, která znamená **W**eb-scale **S**upervised **P**retraining for **S**peech **R**ecognition. Whisper je automatický rozpoznávač řeči, založený na principu modelů typu Transformer, natrénovaný na obrovském množství vícejazyčných a víceúlohových dat. Celkový počet použitých dat pro trénování činil zaokrouhleně 680 000 hodin. Z toho 65 % tvořila anglicky mluvená data, 17 % tvořila data obsahující mluvenou řeč v 96 různých jazycích a 18 % dat představovala data přeložená z těchto jazyků do angličtiny. Takto enormní datová sada pro trénování vznikla spojením velkého množství veřejných datasetů jako jsou např. LibriSpeech[3], CommonVoice[4] nebo VoxPopuli[5]. Kromě přepisu řeči umožňuje Whisper také překládat řeč napříč jazyky. Whisper pro svoji správnou funkci vyžaduje, aby vstupní zvuková data byla navzorkována na frekvenci 16 kHz a současně musí být data převedena do formy 80-ti kanálových Mel-spektrogramů, které jsou generované po oknech s délkou 25 ms a to s kroky o délce 10 ms.

[1] https://pypi.org/project/AudioAugmentor/
[2] https://openai.com/
[3] https://www.openslr.org/12
[4] https://commonvoice.mozilla.org/en
[5] https://huggingface.co/datasets/facebook/voxpopuli

Whisper přidává navíc do vygenerovaných přepisů řeči tzv. tokeny, jež určují různé stavy, které mohou při přepisu nastat. Tyto tokeny jsou přidávány z toho důvodu, aby Whisper rozpoznal jednotlivé požadavky, které musí při rozpoznávání řeči splnit. Tokeny mohou být následující:

- **<|startoftranscript|>** – Tento token určuje moment, kdy má dojít k začátku přepisu řeči.
- **Jazykové tokeny** – Tyto tokeny určují jazyk řeči, která se nachází v dané zvukové nahrávce. Token <|cs|> např. značí výskyt češtiny ve zpracovávané řeči.
- **<|nospeech|>** – Tento token určuje úsek, kde se nenachází žádná řeč.
- **<|transcribe|>** nebo **<|translate|>** – Tyto dva tokeny specifikují, zda má Whisper pouze přepisovat řeč nebo jestli ji má i překládat.
- **<|notimestamps|>** – Tímto tokenem se určí, zda má Whisper generovat časové značky v přepisech řeči.
- **<|endoftranscript|>** – Tento token určuje pouze konec přepisu řeči.

Whisper je možné používat s různými velikostmi svých modelů. Tyto jednotlivé velikosti modelů se primárně liší počtem parametrů, což ovlivňuje množství místa, které jednotlivé modely zaberou na disku, ale i jejich náročnost pro samotné spuštění.

Model	**Počet vrstev**	**Počet parametrů**
Tiny	4	39M
Base	6	74M
Small	12	244M
Medium	24	769M
Large	32	1550M

Tabulka 6.1: Přehled vlastností jednotlivých modelů. Tabulka převzata a upravena z [9].

Whisper je distribuován v několika formách. Jedna z forem je jednoduchá terminálová aplikace, kterou může uživatel spouštět s různými parametry. Další formou je samostatný modul jazyka Python, jež obsahuje jednoduché rozhraní pro jeho použití. Whisper je ale také šířen např. jako součást knihovny Hugging Face Transformers 6. V této práci využívám automatický rozpoznávač řeči Whisper jako nástroj, který se snažím vyladit k lepším výsledkům s využitím mnou vytvořené knihovny pro augmentaci zvukových dat.

Platforma Hugging Face

Online platforma Hugging Face[6] je zaměřená na vývoj a sdílení modelů umělé inteligence a dbá na podporu otevřenosti a spolupráce v oblasti vývoje těchto modelů. Této podpory otevřenosti a spolupráce platforma Hugging Face dosahuje tím, že nabízí na svých webových stránkách portál s názvem „Model Hub“, pomocí kterého mohou komunity vývojářů sdílet své vytvořené modely a jejich metriky, čímž je zpřístupní širší veřejnosti. Platforma Hugging Face nabízí také široké spektrum datových sad, které je možné využít k trénování jednotlivých modelů. Uživatel si ovšem může tyto datové sady vytvořit sám a následně je využít pomocí knihovny Datasets 6.

Dále je také možné prezentovat funkcionalitu jednotlivých nahraných modelů s využitím tzv. prostorů (angl. Spaces). Tato funkcionalita umožňuje vytvářet interaktivní rozhraní, pomocí kterého si uživatelé mohou vyzkoušet různé modely, jež jsou snadno spustitelné v internetovém prohlížeči. Uživatel tedy nepotřebuje podstoupit často složitý proces stahování a nasazování určitých modelů, pokud si chce např. jenom vyzkoušet jeho základní funkcionalitu. Toto nasazení modelů na platformu Hugging Face je pro přihlášeného uživatele zdarma. Takto nahrané modely jsou ovšem spouštěny pouze na procesorech. Pokud by uživatel chtěl, aby jeho modely byly spouštěny na grafických akcelerátorech, musí zaplatit určený hodinový poplatek dle výkonnosti využitého akcelerátoru.

V mé práci využívám tuto platformu pro sdílení modelů automatického rozpoznávače řeči Whisper, které byly vyladěny s použitím augmentačního nástroje implementovaného v rámci této bakalářské práce.

Knihovna Transformers

Hugging Face Transformers[7] je knihovna poskytující rozhraní, které umožňuje stahovat, používat a trénovat velké množství nejmodernějších modelů umělé inteligence. Tuto knihovnu je možné využívat současně s knihovnou PyTorch[8]. Předtrénované modely obsažené v knihovně Transformers dokáží provádět rozmanité druhy úloh nad různými druhy dat. V této práci je knihovna Transformers využita ke stažení a použití předtrénovaných modelů automatického rozpoznávače řeči Whisper 6. Díky jednoduchému rozhraní je použití této knihovny velmi snadné.

Knihovna Datasets

Hugging Face Datasets[9] je knihovna, pomocí které může vývojář s využitím pouhých pár řádků kódu používat tisíce datových sad publikovaných na platformě Hugging Face. Platforma poskytuje různé typy dostupných datových sad. Některé z nich se specializují na detekci objektů z obrázků a videí, zatímco jiné obsahují tabulková nebo zvuková data apod. Knihovna také nabízí nástroje umožňující následnou práci s datovými sadami. Tyto nástroje umožňují např. aplikovat augmentační funkce nad celými datasety, mohou odebírat atributy dat nebo měnit vzorkovací frekvence nahrávek obsažených ve zvukových datových sadách.

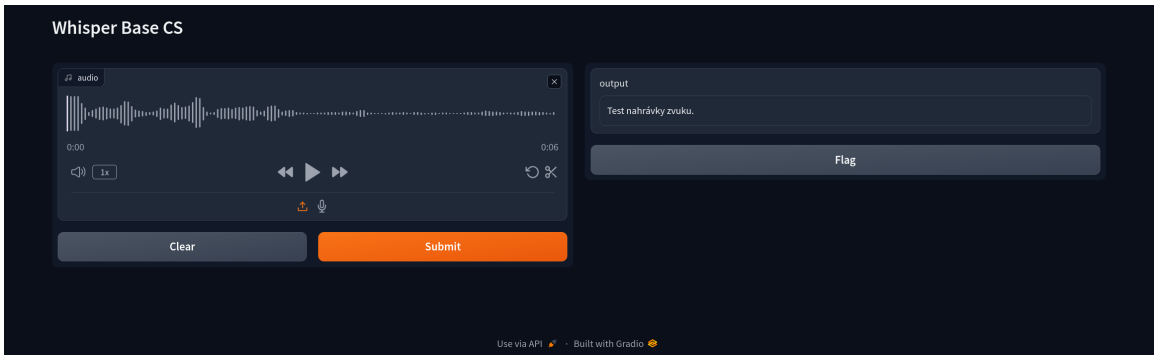
[6] https://huggingface.co/
[7] https://huggingface.co/docs/transformers/
[8] https://pytorch.org/
[9] https://huggingface.co/docs/datasets/

Google Colab

Google Colab[10] je služba umožňující spouštět kód jazyka Python s použitím Jupyter[11] zápisníků přímo ve webovém prohlížeči. Hlavní vlastností služby Google Colab je, mimo možnosti spouštění kódu jazyka Python odkudkoliv a kdykoliv, možnost spouštět tento kód na velmi výkonných grafických akcelerátorech s velkým množstvím VRAM. Konkrétně Colab umožňuje spouštět kód buďto na procesoru nebo na grafických kartách Nvidia T4, Nvidia V100, Nvidia A100 nebo na proprietárních TPU (Tensor Processing Unit) společnosti Google. Výše zmíněné grafické karty ale stojí řádově tisíce amerických dolarů, takže Google Colab nabízí tuto službu zdarma pouze s možností spouštět svůj kód na procesoru nebo na grafické kartě Nvidia T4, jejíž stabilní dostupnost není zaručena. Dále je v neplacené verzi dostupná menší kapacita na discích nebo méně paměti RAM. Pokud chce uživatel stabilní přístup ke grafické kartě a chce také využívat výkonnější grafické karty než je Nvidia T4, musí si nakoupit tzv. výpočetní jednotky. Z nakoupeného množství výpočetních jednotek se dle aktuálně využívané grafické karty odečítá určitý počet kreditů. Čím je grafická karta výkonnější, tím je odečítáno více výpočetních jednotek za hodinu. Např. dnes (2.5.2024) lze nakoupit 100 výpočetních jednotek za 11,19 € a hodinový běh kódu s využitím nejvýkonnější grafické karty Nvidia A100 spotřebuje cca 15 výpočetních jednotek za hodinu. Pro potřeby této práce jsem výše zmíněné možnosti využil, a to pro vyzkoušení ladění větších modelů automatického rozpoznávače řeči Whisper 6, které jsou velmi náročné na množství vyžadované VRAM v grafickém akcelerátoru.

Knihovna Gradio

Gradio[12] je otevřená knihovna jazyka Python navržená tak, aby ulehčila a zrychlila vývoj demonstračních aplikací modelů umělé inteligence, aplikačních rozhraní nebo i obyčejných funkcí. Takovéto demonstrační aplikace je následně velmi triviální vytvořit a nahrát na platformu Hugging Face, kde je možné je veřejně sdílet s ostatními vývojáři.



Obrázek 6.1: Ukázka demo aplikace automatického rozpoznávače řeči Whisper.

[10] https://colab.google/
[11] https://jupyter.org/
[12] https://www.gradio.app/

6.1 Ladění systému Whisper

Funkcionalitu implementované knihovny AudioAugmentor jsem se rozhodl demonstrovat úspěšnou augmentací trénovací a testovací datové sady, které jsou využity při ladění automatického rozpoznávače řeči Whisper 6. Pro ladění jsem si zvolil verzi modelu *Base* a to proto, že je to největší model, který jsem mohl lokálně trénovat na svém zařízení, které má grafickou kartu NVIDIA GeForce RTX 3060 s 6 GB VRAM. Větší modely se již na grafickou kartu nevešly. Tento *Base* model jsem se rozhodl dotrénovat tak, aby byla zlepšena jeho výkonnost oproti českému jazyku a získal jsem ho s využitím knihovny Transformers 6. Jako trénovací a testovací data jsem využil část datové sady CommonVoice verze 11[13], která obsahuje pouze česky namluvené nahrávky. Tuto datovou sadu jsem získal pomocí knihovny *Datasets* 6. V trénovací části se nachází 22 155 nahrávek a v testovací se nachází 7 714 nahrávek.

Jako součást tohoto experimentu jsem se snažil dotrénovat 2 modely *Base* systému Whisper. První model byl dotrénován na trénovací datové sadě bez aplikovaných augmentací a druhý byl dotrénován na datové sadě s již aplikovanými augmentacemi. Každý z těchto modelů byl následně otestován na testovací datové sadě, která obsahovala buďto čisté nahrávky nebo nahrávky s aplikovanými augmentacemi. Na trénovací a testovací datové sady jsem aplikoval s různými pravděpodobnostmi následující augmentace:

- Roztažení časové domény – 20% pravděpodobnost
- Zesílení – 10% pravděpodobnost
- Posunutí základního tónu – 20% pravděpodobnost
- Gausovský šum – 20% pravděpodobnost
- Filtr typu dolní propusť – 10% pravděpodobnost
- MP3 komprese – 20% pravděpodobnost

Tabulka níže 6.2 zobrazuje základní metriku WER (Word Error Rate) modelu *Base* oproti čisté a augmentované testovací datové sadě. U metriky WER platí, že čím je menší hodnota, tím lepší.

	WER čistá testovací sada	**WER augmentovaná testovací sada**
Base Whisper	72,1 %	109,9 %

Tabulka 6.2: Základní metriky modelu *Base* systému Whisper.

Trénování systému Whisper jsem spustil s pomocí třídy Seq2SeqTrainer, která je obsažena v knihovně Transformers a umožňuje s jednoduchým nadefinováním trénovacích parametrů spustit trénování modelů umělé inteligence.

Velikost trénovací dávky dat byla nastavena na hodnotu 8, konfigurační číslo náhodného generátoru čísel bylo 42, rychlost učení (angl. learning rate) byla 10^{-5}, jako metoda pro optimalizaci parametrů neuronové sítě byla vybrána metoda gradientního sestupu Adam.

[13] https://huggingface.co/datasets/mozilla-foundation/common_voice_11_0

Trénování bylo spuštěno na 4000 kroků, což odpovídá necelým 6 epochám. Tabulka níže 6.3 zobrazuje hodnoty WER jednotlivých dotrénovaných modelů oproti testovacím datovým sadám, které obsahovaly buďto čisté nebo augmentované nahrávky.

	WER čistá testovací sada	**WER augmentovaná testovací sada**
Base Whisper dotrénovaný na čisté datové sadě	35,1 %	65,9 %
Base Whisper dotrénovaný na augmentované datové sadě	35,2 %	54,2 %

Tabulka 6.3: Výsledky dotrénování modelu *Base* systému Whisper.

Z těchto výsledku je patrné, že modely dotrénované na datech, která obsahují augmentace, jsou výkonnější oproti augmentovaným datovým sadám než modely, jež byly dotrénovány na čistých datových sadách.

6.2 Replikovatelnost experimentů

Důležitou vlastností nástrojů a knihoven umožňující augmentaci dat je to, aby při nastavení určité hodnoty konfigurující náhodný generátor čísel byly výstupy těchto nástrojů pokaždé stejné. Tato vlastnost následně umožňuje výzkumníkům dosahovat stejných hodnot při opakovaném spouštění stejných experimentů.

Splnění této vlastnosti u implementované knihovny AudioAugmentor jsem zkontroloval několika způsoby. Nejdříve jsem nastavil konfigurátory náhodných čísel na určitou hodnotu a několikrát za sebou spustil trénování modelu automatického rozpoznávače řeči Whisper, kde jsem pro augmentaci datové sady, která byla použita pro trénování, využil knihovnu AudioAugmentor. Výsledky těchto několika běhů trénování byly vždy totožné. Dále jsem vyzkoušel několikrát provést totožnou augmentaci zvukové nahrávky a porovnal výsledné nahrávky mezi sebou. Tyto výsledky byly taktéž totožné.

6.3 Porovnání použití s konkurenčními nástroji

Hlavní motivací pro vytvoření knihovny, která by umožňovala uživateli jednoduše aplikovat větší množství různých augmentací na jednotlivé zvukové nahrávky, je právě absence takovéhoto nástroje. Pokud chce totiž uživatel aplikovat několik různých augmentací, musí většinou použít několik různých knihoven, kde každá má jiné vlastnosti (podporované datové typy, podporovaná zařízení). Tento problém právě řeší implementovaná knihovna AudioAugmentor, která uživateli umožní jednoduše aplikovat široké spektrum, jak jednoduchých, tak složitějších augmentací zvukových nahrávek.

Následující ukázka kódu 6.1 demonstruje aplikaci různých augmentací zvukových dat z knihoven, jež jsou integrovány v rámci knihovny AudioAugmentor. Konkrétně se zde aplikuje změna základního tónu nahrávky, přidání gausovského šumu, filtr typu dolní propusť

a také se aplikuje audio kodek G.711. Ukázka kódu 6.2 zobrazuje využití implementované knihovny AudioAugmentor pro aplikaci stejných augmentací jako v ukázce 6.1. Hlavním rozdílem mezi těmito ukázkami je to, že není potřeba pro aplikaci augmentací zvuku z několika různých knihoven využívat odlišná rozhraní a provádět různé manipulace s daty, které jsou pro interoperabilitu jednotlivých knihoven nutné. Stačí pouze využít knihovnu AudioAugmentor, která poskytne jednotné a jednoduché rozhraní pro aplikaci totožných augmentací.

```
import torchaudio
import torchaudio.io as TIO
import torchaudio.transforms as T
import torch_audiomentations as TA
import audiomentations as AA

signal, fs = torchaudio.load('test.wav')
pitch_shift = T.PitchShift(sample_rate=fs, n_steps=4)
pitch_shifted = pitch_shift(signal)

aa_augment = AA.Compose([
    AA.AddGaussianNoise(min_amplitude=0.05, max_amplitude=0.1, p=1),
])
aa_ready_sample = pitch_shifted.detach().numpy()[0]
aa_augmented = aa_augment(samples=aa_ready_sample, sample_rate=fs)

ta_augment = TA.Compose(
    transforms=[
        TA.LowPassFilter(min_cutoff_freq=500,
                         max_cutoff_freq=600,
                         sample_rate=fs,
                         p=1),
    ]
)
ta_ready_sample = torch.from_numpy(aa_augmented)
ta_ready_sample = ta_ready_sample.unsqueeze(0).unsqueeze(0)
ta_augmented = ta_augment(samples=ta_ready_sample, sample_rate=fs)

pcm = TIO.AudioEffector(format="wav", encoder='pcm_mulaw')
eff_ready_sample = ta_augmented.cpu().squeeze(0).transpose(0, 1)
final= pcm.apply(eff_ready_sample, fs)
```

Zdrojový kód 6.1: Definice a aplikace augmentací s využitím zdrojových knihoven daných augmentací.

```
from AudioAugmentor import core, transf_gen

signal, fs = torchaudio.load('test.wav')
transformations = transf_gen.transf_gen(
        PitchShift={'sample_rate': fs,
                    'n_steps': 4,
                    'p': 1},
        AddGaussianNoise='min_amplitude=0.05, max_amplitude=0.1, p=1',
        LowPassFilter={'min_cutoff_freq': 500,
                       'max_cutoff_freq': 600,
                       'sample_rate': fs,
                       'p': 1},
        pcm_mulaw=True
        )
augment = core.AugmentWaveform(transformations=transformations, device='cpu',
sample_rate=fs)
final = augment(signal.numpy()[0])
```

Zdrojový kód 6.2: Definice a aplikace augmentací s využitím knihovny AudioAugmentor.

6.4 Možnost použití knihovny AudioAugmentor s PyTorch

Knihovna AudioAugmentor poskytuje v modulu `core` třídu `Collator`, kterou je možné využít spolu s třídou knihovny PyTorch `DataLoader`. Toto je možné díky tomu, že je třída `Collator` navržena jako funkce `collate_fn`, která se využívá jako argument třídy `DataLoader` a je zodpovědná za seskupování dat, které třída `DataLoader` poskytuje na svém výstupu. Při tomto seskupování dat jsou právě třídou `Collator` aplikovány augmentace zvukových dat, dle uživatelova zadání.

Následující ukázka kódu 6.3 představuje využití třídy `Collator` spolu s třídou `DataLoader` pro augmentaci zvukové datové sady. Datová sada, která je využita pro augmentaci je v tomto příkladě LibriSpeech[14] a je stažena s využitím funkcí knihovny `torchaudio`[15]. Konkrétně se na datovou sadu aplikují augmentace posunutí základního tónu, změna rychlosti, filtr typu dolní propust, gausovský šum a kodek G.726.

[14] https://pytorch.org/audio/main/generated/torchaudio.datasets.LIBRISPEECH.html
[15] https://pytorch.org/audio/stable/index.html

```python
import torch
import torchaudio
from AudioAugmentor import core, transf_gen

dataset = torchaudio.datasets.LIBRISPEECH(root='', url='train-clean-100', download=True)
fs = 16000

transformations = transf_gen.transf_gen(verbose=True,
    PitchShift={'sample_rate': fs, 'n_steps': [1, 1.5, 0.1], 'p': 1},
    Speed={'orig_freq': fs, 'factor': [0.9, 1.5, 0.1], 'p': 1},
    LowPassFilter={'min_cutoff_freq': 700, 'max_cutoff_freq': 800,
    'sample_rate': fs, 'p': 1},
    AddGaussianNoise={'min_amplitude': 0.001, 'max_amplitude': 0.015, 'p': 1},
    g726={'audio_bitrate': '40k'},
    )

collate_fn = core.Collator(
    transformations=transformations, device='cpu', sample_rate=fs, verbose=True
)

aug_dataloader = torch.utils.data.DataLoader(
    dataset,
    collate_fn=collate_fn,
)

augmented_audio = next(iter(aug_dataloader))
```

Zdrojový kód 6.3: Použití knihovny AudioAugmentor spolu s knihovnou PyTorch.

6.5 Aplikace vlastních impulsních odezev a šumů

Při augmentaci zvukových dat může nastat situace, ve které potřebuje výzkumník použít vlastní impulsní odezvy nebo přidat vlastní šumy či nahrávky do pozadí augmentovaného zvuku. Tyto impulsní odezvy mohou být např. naměřené reálné impulsní odezvy místností. Nahrávky, které mohou být využity pro vložení do pozadí zvuku mohou pocházet např. z datové sady MUSAN (Music Speech And Noise corpus) [13].

Pro aplikaci vlastních impulsních odezev pomocí knihovny AudioAugmentor musí výzkumník využít augmentaci nazvanou `ApplyImpulseResponse`. Této funkci poskytne výzkumník jako argument parametru *ir_paths* cestu k adresáři s impulsními odezvami nebo seznam jednotlivých cest k impulsním odezvám.

Funkce `AddBackgroundNoise` aplikuje přidání nahrávky či šumu do pozadí zvuku a využívá podobné rozhraní jako výše zmíněná funkce `ApplyImpulseResponse`. Výzkumník musí jako argument parametru *background_paths* poskytnout cestu k adresáři s nahrávkami nebo seznam s cestami nahrávek, které chce aplikovat do pozadí právě augmentovaného zvuku. Ukázka kódu níže 6.4 představuje využití těchto funkcí.

```
from AudioAugmentor import core, transf_gen

rir_dir='rirDirectory/'
noise_rir='noiseDirectory/'
signal, fs = torchaudio.load('test.wav')
transformations = transf_gen.transf_gen(
        ApplyImpulseResponse=f'''ir_paths={rir_dir},
                                 p=1,
                                 sample_rate={fs}''',
        AddBackgroundNoise=f'''background_paths={noise_dir},
                        min_snr_in_db=10,
                        max_snr_in_db=20,
                        p=1,
                        sample_rate={fs}''',
        )
augment = core.AugmentWaveform(transformations=transformations, device='cpu',
sample_rate=fs)
final = augment(signal.numpy()[0])
```

Zdrojový kód 6.4: Použití vlastních impulsních odezev a šumů při augmentaci zvukových dat s použitím knihovny AudioAugmentor.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat nástroj, který bude integrovat různorodé augmentace zvukových nahrávek tak, aby byl tento nástroj využitelný spolu s knihovnou PyTorch.

Pro vytvoření vhodného návrhu jsem si nastudoval funkcionalitu knihovny PyTorch a také jsem prozkoumal již existující nástroje a knihovny umožňující augmentaci zvukových dat. Na základě porovnání těchto dostupných nástrojů jsem vybral ty nejvhodnější a zakomponoval je do vytvořené knihovny jazyka Python nazvané AudioAugmentor, která poskytuje jednoduché rozhraní nad těmito integrovanými nástroji. AudioAugmentor lze použít nejen pro augmentaci zvukových datových sad s využitím knihovny PyTorch, ale umožňuje také augmentaci samotných zvukových nahrávek či lokálních datových sad. Knihovna AudioAugmentor je veřejně publikována na platformě PyPi, kde si ji může kdokoliv jednoduše stáhnout a okamžitě začít využívat.

Implementovaná knihovna byla použita pro augmentaci trénovacích a testovacích datových sad, které byly využity v rámci ladění systému pro automatický přepis řeči Whisper za účelem zlepšení jeho výkonnosti na českém jazyce. Dále byla knihovna AudioAugmentor úspěšně otestována, zda je s ní možné replikovat experimenty.

Tato práce mi dala dobrý vhled nejen do knihovny PyTorch, ale i do procesů augmentace dat, které se využívají při trénování modelů umělé inteligence. V budoucnu bych chtěl tyto poznatky určitě využít při své další práci s různými modely umělé inteligence.

Následná práce na knihovně AudioAugmentor by mohla zahrnovat integraci bakalářské práce vypracovávané taktéž na ÚPGM FIT VUT, která se zabývá simulací přenosu řeči pomocí VHF kanálu, což by mohla být jedna z dalších podporovaných augmentací. Dále by mohla být přidána funkcionalita simulace libovolně se pohybujícího mluvčího uvnitř místnosti. Tuto práci jsem také prezentoval na konferenci Excel@FIT 2024.

Literatura

[1] *PyTorch documentation* [online]. ©2023 [cit. 2023-10-25]. Dostupné z: https://pytorch.org/docs/stable/index.html.

[2] *Torchaudio documentation* [online]. ©2023 [cit. 2023-10-25]. Dostupné z: https://pytorch.org/audio/stable/.

[3] Allen, J. a Berkley, D. Image method for efficiently simulating small-room acoustics. *The Journal of the Acoustical Society of America*. Duben 1979, sv. 65, s. 943–950. DOI: 10.1121/1.382599.

[4] Bittner, R. M., Humphrey, E. J. a Bello, J. P. PySOX: Leveraging the Audio Signal Processing Power of SOX in Python. In: *Proceedings of the 17th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers, New York City, USA*. Srpen 2016 [cit. 2023-10-26]. Dostupné z: https://api.semanticscholar.org/CorpusID:64909709.

[5] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G. et al. *Deep Speech: Scaling up end-to-end speech recognition*. 2014 [cit. 2023-10-17].

[6] Leitner, B. Z. J. a Thornton, S. Audio Recognition using Mel Spectrograms and Convolution Neural Networks. In:. 2019 [cit. 2023-10-16]. Dostupné z: https://api.semanticscholar.org/CorpusID:237274283.

[7] Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B. et al. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In: *Interspeech 2019*. ISCA, Září 2019 [cit. 2023-10-16]. DOI: 10.21437/interspeech.2019-2680. Dostupné z: https://doi.org/10.21437%2Finterspeech.2019-2680.

[8] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Buc, F. d'Alché, Fox, E. et al., ed. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, s. 8024–8035. Dostupné z: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[9] Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C. et al. *Robust Speech Recognition via Large-Scale Weak Supervision*. 2022 [cit. 2024-01-03]. Dostupné z: https://arxiv.org/abs/2212.04356.

[10] Salman, S. a Liu, X. Overfitting Mechanism and Avoidance in Deep Neural Networks. *ArXiv*. 2019, abs/1901.06566, [cit. 2023-10-16]. Dostupné z: https://arxiv.org/abs/1901.06566.

[11] Scheibler, R., Bezzam, E. a Dokmanic, I. Pyroomacoustics: A Python Package for Audio Room Simulation and Array Processing Algorithms. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Duben 2018 [cit. 2023-10-26]. DOI: 10.1109/icassp.2018.8461310. Dostupné z: https://doi.org/10.1109%2Ficassp.2018.8461310.

[12] Schlüter, J. a Grill, T. Exploring Data Augmentation for Improved Singing Voice Detection with Neural Networks. In: *International Society for Music Information Retrieval Conference*. 2015 [cit. 2023-10-17]. Dostupné z: https://api.semanticscholar.org/CorpusID:3999220.

[13] Snyder, D., Chen, G. a Povey, D. *MUSAN: A Music, Speech, and Noise Corpus*. 2015. ArXiv:1510.08484v1.

[14] Steinberg, J. C. Positions of Stimulation in the Cochlea by Pure Tones. *The Journal of the Acoustical Society of America*. Leden 1937, sv. 8, č. 3, s. 176–180. DOI: 10.1121/1.1915891. ISSN 0001-4966. Dostupné z: https://doi.org/10.1121/1.1915891.

[15] Szőke, I., Skácel, M., Mošner, L., Paliesek, J. a Černocký, J. Building and evaluation of a real room impulse response dataset. *IEEE Journal of Selected Topics in Signal Processing*. 2019, sv. 13, č. 4, s. 863–876, [cit. 2023-10-17]. DOI: 10.1109/JSTSP.2019.2917582.

[16] Wei, S., Zou, S., Liao, F. a lang weimin. A Comparison on Data Augmentation Methods Based on Deep Learning for Audio Classification. *Journal of Physics: Conference Series*. IOP Publishing. Leden 2020, sv. 1453, č. 1, s. 012085, [cit. 2023-10-17]. DOI: 10.1088/1742-6596/1453/1/012085. Dostupné z: https://dx.doi.org/10.1088/1742-6596/1453/1/012085.

Příloha A

Obsah přiloženého paměťového média

```
/
├── xvasin11-plakat.pdf ..... Náhled přiložen jako příloha B
├── xvasin11-BP.pdf ..... Text práce odevzdané do IS VUT
├── xvasin11-BP-tisk.pdf ..... Verze textu práce pro tisk
├── tex/ ..... Adresář se zdrojovými soubory textu práce
├── src/ ..... Veškeré zdrojové kódy implementované knihovny
└── README.md ..... Návod k implementované knihovně
```

Příloha B

Plakát