VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2024 Jerguš Čermák

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

REVERZNÍ INŽENÝRING MIXÁŽE POMOCÍ NEURONOVÉ SÍTĚ

REVERSE ENGINEERING OF AN AUDIO MIX USING NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Jerguš Čermák

AUTHOR

VEDOUCÍ PRÁCE — prof. Mgr. Pavel Rajmic, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Audio inženýrství**
specializace Zvuková produkce a nahrávání
Ústav telekomunikací

Student: Jerguš Čermák ***ID:*** 240145
Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Reverzní inženýring mixáže pomocí neuronové sítě

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte koncept Differential Digital Signal Processing (DDSP) [1] a seznamte se s lineárními prvky, které tato knihovna nabízí, jako jsou gain, EQ, reverb, panning. Pomocí seřazení těchto prvků jako v mixážním pultu realizujte několik mixů z předem připravených čistých zvukových stop.

Vytvoření mixáže chápejte jako provedení výpočtu neuronové sítě se speciální strukturou, na jejímž vstupu jsou stopy a na výstupu výsledný mix. Použijte techniku učení parametrů sítě (tedy parametrů EQ, reverbu apod.) představenou v [2] k odhadu původních parametrů použitých při mixáži.

Přesnost odhadu srovnejte objektivně a subjektivně (zpětnou remixáží a poslechovým srovnáním původního a odvozeného mixu).

DOPORUČENÁ LITERATURA:

[1] Engel, J., Hantrakul, L., Gu, C., Roberts, A. DDSP: Differentiable Digital Signal Processing. Proceedings of ICLR 2020.
[2] Colonel, J. T., Reiss, J. Reverse engineering of a recording mix with differentiable digital signal processing. The Journal of the Acoustical Society of America, 2021.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 28.5.2024

Vedoucí práce: prof. Mgr. Pavel Rajmic, Ph.D.

doc. Ing. Jiří Schimmel, Ph.D.
předseda rady studijního programu

UPOZORNĚNÍ:
Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Bakalárska práca sa zaoberá využitím algoritmov neurónovej siete za účelom zistenia parametrov signálových procesorov použitých pri mixáži zvukových stôp. V rámci práce sú prezentované lineárne signálové procesory *Gain*, *Pan*, *Filter* a *Reverb* umožňujúce úpravu zvukového signálu a vytvorenie stereofónneho mixu zvukovej nahrávky. Následne sú vďaka implementácií pomocou knižnice DDSP použité v zmysle vrstiev modelu neurónovej siete, ktorý je zameraný na predikciu parametrov použitých pri mixáži, za predpokladu znalosti vstupných stôp a cieľového mixu. V rámci práce boli vytvorené stereofónne mixy, ktorých parametre boli následne odhadované pomocou dvoch modelov neurónovej siete. Výsledky boli posudzované ako objektívnymi, tak subjektívnymi metódami (posluchovým testom).

KĽÚČOVÉ SLOVÁ

Neurónové siete, DDSP, LTI zvukové efekty, stereofónna mixáž

ABSTRACT

This bachelor's thesis focuses on the use of neural network algorithms to determine the parameters of signal processors used in the mixing of audio tracks. The thesis presents linear signal processors such as *Gain*, *Pan*, *Filter*, and *Reverb*, which are commonly used to process audio signal and to produce a stereo mix of the audio recording. These processors are subsequently used within the neural network model as layers, implemented using the DDSP library, aimed at predicting the parameters used in the mix, given the knowledge of the input tracks and the target mix. Resultantly, stereo mixdowns were created, and their parameters were estimated using two neural network models. The results were evaluated using both objective measurements and subjective methods (listening test).

KEYWORDS

Neural networks, DDSP, LTI audio effects, stereo mixdown

ČERMÁK, Jerguš. *Reverzní inženýring mixáže pomocí neuronové sítě*. Bakalárska práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedúci práce: prof. Mgr. Pavel Rajmic, PhD.

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Jerguš Čermák
VUT ID autora:	240145
Typ práce:	Bakalárska práca
Akademický rok:	2023/24
Téma závěrečnej práce:	Reverzní inženýring mixáže pomocí neuronové sítě

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Chcel by som sa poďakovať vedúcemu svojej práce pánu prof. Mgr. Pavlovi Rajmicovi, Ph.D. za odborný prístup, konzultácie, trpezlivosť, nápomocnosť a podnetné návrhy k práci.

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

Tvorba zvukovej nahrávky v dnešnej dobe obnáša oveľa viac, ako len nahrávanie hudobného výkonu. Nie je to výsledok práce umelcov, je to výsledok spolupráce hudobníkov so zvukovými inžiniermi, ktorý výslednú nahrávku výrazne ovplyvňujú v procese mixáže a masteringu. V procese mixáže je nahrávka upravovaná radou audio efektov, pomocou ktorých sme schopný korigovať hlasitosť jednotlivých nahraných stôp, ich frekvenčné spektrum, či umiestnenie a rozprestrenie v priestore. Tieto úpravy slúžia na zlúčenie jednotlivých nahraných stôp do výslednej nahrávky, v ktorej sú nahrané signály spracované s ohľadom na umelecké a technické požiadavky [1] .

Pri mixáži zvuku používa zvukový inžinier signálové procesory, označované tiež efekty, ktorých použité parametre ovplyvňujú zvuk výsledného mixu. Tieto efekty sú aplikované na jednotlivých stopách (môžu byť aplikované aj na výslednej stereofónnej stope, tzv. *Master*) a ich parametre sú špecificky použité v danom konkrétnom mixe. Pokiaľ neexistuje záznam, či už písomný alebo multimediálny, z mixážneho procesu, je na základe výslednej nahrávky a vstupných stôp takmer nemožné dohľadať parametre jednotlivých procesorov na dosiahnutie výsledného mixu.

Práca sa zaoberá využitím knižnice DDSP pri reverznom inžinieringu mixáže zvukových stôp. Základom práce je hľadanie parametrov jednotlivých efektov použitých pri mixáži zvukového diela. Tieto efekty sú diferencovateľné (použité z knižnice *DDSP*) a použité v zmysle vrstiev neurónovej siete. Parametre jednotlivých signálových procesorov sú teda hľadané použitím algoritmov strojového učenia ako váhy neurónovej siete. Použitými efektami budú FIR filter (filter s konečnou impulznou odozvou), Reverb (dozvuk), Gain (efekt ovplyvňujúci hlasitosť) a Pan (efekt ovplyvňujúci stereofónny obraz nahrávky). Knižnica DDSP neimplementuje nelineárne efekty na úpravu dynamiky signálu (kompresory, skreslenie).

Cieľom práce je zostaviť funkčný model neurónovej siete, založený na signálových procesoroch implementovaných pomocou knižnice *DDSP*, ktorý by bol schopný slúžiť k odhadu parametrov použitých pri mixáži a následne presnosť odhadu porovnať objektívnymi a subjektívnymi metódami. Čiastočným cieľom práce je pomocou signálových procesorov implementovaných v *DDSP* vytvoriť mixy slúžiace k následnému učeniu zostaveného modelu neurónovej siete.

Posledným vytýčeným cieľom práce je objektívne a subjektívne porovnanie výsledkov dosiahnutých pomocou implementovaných modelov neurónovej siete.

V rámci teoretických východísk práce sú v prvej kapitole popisované všeobecné princípy neurónových sietí, ich štruktúra, základné jednotky, prepojenia v neurónových sieťach. V rámci podkapitoly Proces učenia neurónovej siete je popísaný proces učenia neurónovej siete so zameraním na algoritmus spätného šírenia (*Backpropaga-*

tion), ktorý bude využívaný pri zisťovaní parametrov efektov použitých v mixe.

Druhá kapitola sa zaoberá implementáciou efektov prostredníctvom knižnice *DDSP*, so zameraním na signálové procesory *Gain*, *Pan*, *Filter* a *Reverb*, ktoré sú následne využité v zmysle vrstiev neurónovej siete za účelom odhadu parametrov použitých pri mixáži zvukového diela. V rámci kapitoly je popísané fungovanie knižnice *DDSP*, jej štruktúra a signálové procesory využívané ďalej v tejto práci.

V tretej kapitole je vysvetlená mixáž zvuku s využitím signálových procesorov implementovaných pomocou knižnice *DDSP*, pričom predstavuje kompletný postup akým vznikali mixáže použité ako cieľové mixy pri učení neurónovej siete.

Štvrtá kapitola práce popisuje štruktúru a praktickú implementáciu neurónovej siete slúžiacej na hľadanie parametrov použitých pri mixáži a následnú remixáž s parametrami určenými v rámci tréningového cyklu.

V záverečnej kapitole je popísané vyhodnotenie a porovnanie výstupov neurónovej siete s cieľovými údajmi, teda ako s cieľovými mixami, tak s určovanými parametrami. Porovnávanie je vykonané ako objektívnymi metódami, tak demonštratívnym subjektívnym posluchovým testom, založeným na porovnávaní výstupných zvukových dát medzi sebou.

1 Neurónové siete

1.1 Princípy neurónovej siete

Pojmom neurónová sieť označujeme výpočetný model, ktorý je niektorými vlastnosťami podobný mozgu. Základným spôsobom uchovávania informácii v neurónovej sieti je nastavenie váh medzi jednotlivými jednotkami. Prestavovanie týchto váh predstavuje základný spôsob učenia neurónovej siete. Vlastnosti neurónovej siete sú určené jej štruktúrou, napríklad počtom neurónov, počtom vrstiev a typom prepojení medzi vrstvami [2].

Základným stavebným prvkom neurónovej siete je neurón. Neurón je matematická funkcia, do ktorej vstupuje jedna alebo viac hodnôt a jej výsledkom je jedna číselná hodnota. Matematicky sa dá neurón definovať vzťahom:

$$y = f(\sum_i x_i w_i + b)$$

kde f je aktivačná funkcia, x_i je i-tý vstup neurónu, w_i je i-tá váha, ktorou sa násobí i-tý vstup neurónu a b je takzvaný prah [3].

Váha je číselná hodnota, ktorá vyjadruje, do akej miery daný vstup ovplyvňuje výstup neurónu. Vo svojej podstate škáluje (zosilňuje alebo zoslabuje) vstupný signál daného neurónu v sieti. [2]. Vážený súčet $\sum_i x_i w_i$ sa označuje ako aktivačná hodnota [3].

Prah (po anglicky bias) by bolo možné popísať ako konštantu pripočítanú k aktivačnej hodnote za účelom posunutia aktivačnej funkcie [4]. Pripočítaním konštanty b k aktivačnej hodnote je zaistený posun aktivačnej funkcie v kladnom, resp. zápornom smere, čím prakticky nastavuje rozsah vstupných hodnôt aktivačnej funkcie neurónu. V praxi sa prah modeluje ako jedna z váh, ktorá ovplyvňuje vstup rovný hodnote 1 a hodnota váhy je rovná hodnote prahu. Tento prístup zabezpečuje prestavenie prahu v procese učenia [2].

Aktivačná funkcia je matematická funkcia riadiaca správanie neurónu. Určuje jeho výstup pri danej vstupnej hodnote, ktorý nasledovne slúži buď ako vstup do ďalšej vrstvy alebo ako výstup modelu. Aktivačnou funkciou môže byť prakticky akákoľvek matematická funkcia, pri použití učenia pomocou spätného šírenia (anglicky backpropagation) je nutné, aby bola aktivačná funkcia diferencovateľná [5].

Jednou zo základných používaných aktivačných funkcií je lineárna aktivácia, definovaná ako: $y = \sum_i x_i w_i + b$. Táto aktivačná funkcia prenáša aktivačnú hodnotu na výstup nezmenenú. Pri použití tejto aktivačnej funkcie sa sieť skladá z lineárnych funkcií, v dôsledku čoho je model stále lineárnou funkciou vstupov siete, čím sa sieť stáva ekvivalentná lineárnej regresií, ktorá nie je vhodná na riešenie komplexnejších problémov [3].

Z vyššie uvedeného dôvodu sa v neurónových sieťach ako aktivačné funkcie často využívajú nelineárne funkcie. Takouto funkciou je napríklad logistická funkcia alebo sigmoida, definovaná ako: $f(x) = \frac{1}{1+e^{-x}}$. Táto funkcia normalizuje výstup na rozsah hodnôt (0, 1) a využíva sa pri klasifikačných úlohách [5].

Ďalšou často používanou aktivačnou funkciou je hyperbolický tangens tanh, definovaný: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$. Výstup tejto funkcie je normalizovaný v intervale hodnôt $(-1, 1)$. Výhodou funkcie tanh je, že sa ľahšie vysporiadava s negatívnymi hodnotami [2].

Okrem vyššie spomenutých funkcií je často používanou aktivačnou funkciou funkcia **ReLu** (Rectified linear activation function), ktorá je definovaná: $f(x) = \max(0, x)$. Funkcia vracia hodnotu x ak je $x > 0$ a hodnotu 0 v ostatných prípadoch [2].

1.2 Viacvrstvové dopredné neurónové siete

Jedným zo všeobecne používaných modelov architektúry neurónovej siete je *viacvrstvová dopredná neurónová sieť*. V takejto sieti sú neuróny usporiadané do vyšších celkov, takzvaných vrstiev, pričom sieť obsahuje jednu vstupnú vrstvu, jednu alebo viacero takzvaných skrytých vrstiev a jednu výstupnú vrstvu. Neuróny v jednej vrstve zvyčajne používajú identickú aktivačnú funkciu, pričom neuróny v rôznych vrstvách môžu využívať rozličné aktivačné funkcie. Vstupom do vyššej vrstvy neurónovej siete sú výstupy jej predošlej vrstvy [2].

Prvou vrstvou v architektúre siete je vstupná vrstva, pomocou ktorej sa vstupné dáta predávajú neurónovej sieti. Vstupná vrstva býva zvyčajne plne prepojená s neurónmi nasledujúcej skrytej vrstvy, v niektorých štruktúrach však prepojená byť nemusí.

Poslednou vrstvou neurónovej siete je výstupná vrstva, ktorá predstavuje odpoveď, resp. odhad použitého modelu. Predstavuje výstup založený na daných vstupných parametroch predaných na vstupnej vrstve. V závislosti na štruktúre siete môže byť reálna hodnota (v prípade regresného problému), alebo sada pravdepodobností (v prípade klasifikačného problému).

Medzi vstupnou a výstupnou vrstvou sa v topológii siete nachádzajú skryté vrstvy. Skryté vrstvy sú v podstate jadrom neurónovej siete. V tejto časti sa aplikujú aktivačné funkcie na aktivačnú hodnotu a výsledok je ďalej posúvaný ďalším vrstvám. Váhy na jednotlivých prepojeniach uchovávajú informácie získané zo vstupných dát. Obvykle sa každý uzol skrytej vrstvy prepája s každým uzlom vyššej vrstvy. V takom prípade sieti hovoríme, že je plne prepojená [2]. Proces, v ktorom zo vstupov neurónovej siete dostávame výstup sa označuje ako dopredné šírenie (anglicky *forward propagation*) Nastavovanie váh prepojení je základným prvkom

v procese strojového učenia neurónovej siete, ktorým ovplyvňujeme výslednú chybu predikcie od skutočnej hodnoty [6].

1.3 Proces učenia neurónovej siete

Základným cieľom neurónovej siete je modelovať komplexné vzťahy v dátach, za účelom vytvorenia čo najpresnejších predikcií, čo najpresnejšej klasifikácie alebo identifikácie vzorcov vo vstupných dátach [2]. Na základe toho, či poznáme alebo nepoznáme reálnu hodnotu, ku ktorej sa má predikcia modelu blížiť rozlišujeme dva druhy strojového učenia a to učenie s učiteľom a bez učiteľa [7].

Pri učení s učiteľom sa porovnáva predikcia algoritmu so známou reálnou hodnotou (anglicky *target*). Učenie s učiteľom používa takzvané označené dáta (anglicky *labeled data*), čo znamená, že každý tréningový príklad má pridelenú výstupnú hodnotu alebo cieľ. Učenie s učiteľom sa môže rozdeliť na dva typy problémov, a to klasifikáciu a regresiu. V prípade klasifikácie je cieľom siete priradiť každému vstupu správnu triedu alebo kategóriu na základe naučených vzťahov v trénovacej množine [8]. V prípade regresie sa snaží sieť nastaviť parametre tak, aby sa výstupná predikcia čo najviac zhodovala s cieľovými dátami. [7].

V prípade učenia bez učiteľa nepoznáme skutočnú hodnotu, algoritmus hľadá súvislosti a závislosti medzi atribútmi [5]. Pri učení bez učiteľa sa algoritmy snažia nájsť vzory v dátach, ktoré nie sú vopred označené, teda nie je podstatou priblížiť výstup siete cieľovému výstupu. Použitie tohto typu učenia je vhodné pre prípady, kedy chceme pomocou neurónovej siete určiť štruktúru alebo súvislosti medzi dátami [7].

1.3.1 Stratová funkcia

Vzhľadom na to, že každá neurónová sieť je aproximáciou funkcie, predikcia nebude identická ako očakávaná funkcia, bude sa líšiť o hodnotu, ktorú označujeme ako chyba alebo strata. Hlavným cieľom procesu učenia je minimalizovať túto chybu s ohľadom na jednotlivé váhy v sieti. Chybu neurónovej siete vieme vhodne kvantifikovať správne zvolenou stratovou funkciou. Okrem toho táto funkcia zabezpečuje, že odchýlka predikcie siete od skutočného výsledku bude vyjadrená jedným číslom [3]. Učenie neurónovej siete sa dá popísať ako hľadanie parametrov siete (váhy a prahy) za účelom minimalizácie chyby vyjadrenej stratovou funkciou. Vo väčšine prípadov nie je možné nájsť tieto parametre analyticky, ale obvykle je možné aproximovať parametre pomocou iteratívnych algoritmov (napr. zostup gradientu). V závislosti na povahe riešeného problému sa chyba kvantifikuje rozličnými stratovými funkciami [3]. Nakoľko sa reverzný inžiniering mixáže dá považovať za regresný

problém, popíšeme často používané stratové funkcie pre regresné problémy.

Používané stratové funkcie pre regresné problémy

Pre regresné problémy sa používa niekoľko stratových funkcií, najčastejšie používanou je *stredná kvadratická odchýlka* (anglicky *Mean Squared Error*) [2]. Tá je matematicky definovaná vzťahom:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

kde N je celkový počet vzoriek, z ktorých chybu počítame, y_i je skutočná hodnota i-tého vzorku a $\hat{y}_i$ je hodnota predikcie i-tého vzorku neurónovou sieťou. Hodnota chyby vypočítaná pomocou strednej kvadratickej odchýlky je vždy nezáporná vzhľadom na umocnenie rozdielu medzi skutočnou a predikovanou hodnotou [2].

Ďalšou často používanou stratovou funkciou je *stredná absolútna chyba* (anglicky *Mean Absolute Error*). Je definovaná vzťahom:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Ďalšími často používanými stratovými funkciami sú *stredná kvadratická logaritmická odchýľka* (anglicky *Mean Squared Logarithmic Error*), prípadne *stredná absolútna percentuálna odchýľka* (anglicky *Mean absolute percentage error*) [2]. V tejto bakalárskej práci je využitá ako stratová funkcia *stredná kvadratická odchýľka.*

1.3.2 Algoritmus spätného šírenia: Backpropagation

Cieľom optimalizácie neurónovej siete je, aby sa prestavovaím váh v sieti minimalizovala odchýľka odhadu siete od predpokladanej hodnoty vyjadrenej prostredníctvom stratovej funkcie [6]. Za týmto účelom sa používa algoritmus spätného šírenia (anglicky *Backpropagation*).

Minimum stratovej funkcie sa pri neurónových sieťach hľadá pomocou *zostupu gradientu*, čo je iteratívna metóda, pri ktorej sa postupne upravujú jednotlivé váhy za účelom minimalizácie straty siete. Matematicky sa princíp gradientného zostupu dá formulovať:

$$w_i = w_{i-1} - \alpha \cdot \frac{\partial \mathrm{E}}{\partial w_i}$$

kde w_i predstavuje i-té nastavenie váhy, w_{i-1} nastavenie váhy na konci poslednej iterácie, α predstavuje konštantu vyjadrujúcu veľkosť zmeny v závislosti na hodnote derivácie počas jednej iterácie (v praxi označovaná ako *learning rate*), E predstavuje

stratovú funkciu a $\frac{\partial E}{\partial w_i}$ predstavuje parciálnu deriváciu stratovej funkcie s ohľadom na prestavovanú váhu.

Zo vzťahu vyplýva, že váha je postupne prepočítaná počas jednotlivých iterácií. Cieľom algoritmu je pomocou nastavovania váh nájsť minimum stratovej funkcie, kedy bude rozdiel medzi odhadom siete a skutočnou hodnotou minimálny [3].

Optimalizačný algoritmus Adam

Adam je označenie pre adaptívny odhad momentu, ktorý kombinuje dva používané druhy optimalizácie založené na zostupe gradientu, a to momentovú optimalizáciu a *RMSprop* optimalizáciu.

Momentová optimalizácia je založená na princípe použitia kĺzavého priemeru gradientov vypočítaných v uplynulých iteráciách [9]. Tento prístup prináša rýchlejšiu konvergenciu algoritmu oproti zostupu gradientu bez momentu, pri ktorej je váha prestavovaná len na základe aktuálnej hodnoty gradientu v danej iterácií [10]. Momentová optimalizácia pri výpočte aktualizácie váhy zohľadňuje aktualizáciu váhy použitú v predošlej iterácií podľa vzťahu:

$$m_i = \beta \cdot m_{i-1} - \alpha \cdot g_i$$

kde m_i predstavuje aktualizáciu váhy i-tej iterácií, β predstavuje koeficient, ktorý určuje, akú časť aktualizácie váhy z predchádzajúcej iterácie chceme zahrnúť v súčasnej aktualizácií (číslo v intervale $0, 1$), α je konštanta určujúca vplyv gradientu vypočítaného v aktuálnej iterácií na aktualizáciu váhy a g_i vyjadruje vypočítaný gradient s ohľadom na aktualizovanú váhu [10].

Aktualizovanú váhu vyjadríme ako:

$$w_i = w_{i-1} + m_i$$

Druhým princípom v optimalizátore *Adam* je využitie *RMSprop* optimalizácie. Skratka *RMSProp* označuje *Root Mean Square propagation*, kde *Root Mean Square* označuje kvadratický priemer [11], nakoľko je metóda založená na priemerovaní druhých mocnín vypočítaných gradientov. Matematický princíp aktualizácie váhy môže byť vyjadrený ako:

$$v_i = \beta \cdot v_{i-1} + (1 - \beta) \cdot g_i^2$$

kde v_i predstavuje aktualizáciu váhy v i-tej iterácií, β je koeficient určujúci akú časť aktualizácie s predchádzajúcej iterácie zohľadňujeme v aktuálnej iterácií, a g_i^2 je druhá mocnina gradientu vypočítaného s ohľadom na aktualizovanú váhu [10].

Prístup využitý v *RMSprop* je založený na použití exponenciálne váženého kĺzavého priemeru druhých mocnín gradientov. Vo výpočte aktualizácie váhy sú tak

výraznejšie zohľadnené vypočítané aktualizácie z nedávnych iterácií, pričom vplyv dávno uplynulých iterácií sa znižuje v závislosti na nastavení parametru β) [10].

Váha sa pri použití algoritmu *RMSprop* aktualizuje podľa vzťahu:

$$w_i = w_{i-1} - \frac{\alpha \cdot g_i}{\sqrt{v_i + \epsilon}}$$

kde, w_i je aktualizovaná váha, w_{i-1} označuje váhu pred aktualizáciou, α predstavuje *learning rate*, g_i je gradient vypočítaný v aktuálnej iterácií, v_i je koeficient aktualizácie váhy vypočítaný podľa predošlého vzťahu a ϵ je prvok zaisťujúci, že nedôjde k deleniu nulou (typicky používaná hodnota je 10^{-10}) [12].

Optimalizačný algoritmus *Adam* kombinuje prístup momentovej optimalizácie a optimalizácie *RMSprop*, pričom obdobne ako momentová optimalizácia uvažuje gradienty s predchádzajúcej iterácie a ako *RMSprop* zahŕňa druhú mocninu predchádzajúcej iterácie [10]. Matematicky by sa aktualizácia váhy dala zhrnúť ako:

$$g_i = \frac{\partial \mathrm{E}}{\partial w_i}$$

$$m_i = \beta_1 \cdot m_{i-1} + (1 - \beta_1) \cdot g_i$$

$$v_i = \beta_2 \cdot v_{i-1} + (1 - \beta_2) \cdot g_i^2$$

$$m_i = \frac{m_i}{1 - \beta_1^i}$$

$$v_i = \frac{v_i}{1 - \beta_1^i}$$

$$w_i = w_{i-1} - \frac{\alpha \cdot m_i}{\sqrt{v_i + \epsilon}}$$

V prvom kroku sa vypočíta gradient g_i chyby E podľa aktualizovanej váhy w_i. Z tohto gradientu sa vypočíta koeficient m_i, označovaný ako *prvý moment*. Počiatočná hodnota koeficientu m_0 je 0. Následne sa vypočíta koeficient, označovaný ako *druhý moment*, v_i. Počiatočná hodnota koeficientu v_0 je taktiež 0. Následne sa oba koeficienty vydelia za účelom odstránenia chyby spôsobenej inicializáciou parametrov m a v nulovou hodnotou. Následne je v poslednom kroku je vypočítaná aktualizovaná váha, podobne ako v prípade *RMSprop*. Parameter α označuje *learning rate* [13].

Optimalizátor *Adam* je použitý v tejto bakalárskej práci ako optimalizačný algoritmus pri reverznom inžinieringu mixáže.

2 Differentiable digital signal processing: DDSP

2.1 Knižnica DDSP

2.1.1 Princíp knižnice DDSP

Knižnica DDSP je knižnica napísaná v jazyku Python, umožňujúca kombinovať štandardné prístupy používané v digitálnom spracovaní signálu (DSP) s prístupmi takzvaného hlbokého učenia. Je postavená na základe platformy *TensorFlow*, ktorá je špecializovaná na implementáciu algoritmov strojového učenia. Skratka DDSP vyjadruje názov *Differentiable Digital Signal Processing*, v preklade *Diferencovateľné digitálne spracovanie signálu*. Knižnica je založená na použití signálových procesorov bežne používaných v digitálnom spracovaní signálu, napríklad syntezátorov, filtrov či dozvuku, v ich diferencovateľných verziách [14]. Zabezpečením diferencovateľnosti, ktorá je základným predpokladom pre výpočet gradientného zostupu a je predpokladom využitia optimalizačných algoritmov pri použití neurónových sietí, jednotlivých procesorov je dosiahnutá možnosť využitia týchto efektov v algoritmoch hlbokého učenia ako súčasť neurónových sietí.

DDSP je knižnica primárne určená na spracovanie zvukového signálu a je súčasťou projektu *Magenta*, ktorý sa zaoberá využitím algoritmov strojového učenia v hudbe a spracovaní zvuku. Kľúčovým princípom knižnice je využitie jednoducho interpretovateľných prvkov digitálneho spracovania signálu ako prvkov integrovaných do neurónovej siete. Knižnica je dostupná ako projekt s otvoreným zdrojovým kódom [14].

2.1.2 Súčasti knižnice DDSP

Knižnica pozostáva z dvoch primárnych častí. Hlavnou je základná knižnica (*ddsp/*) obsahujúca nástroje potrebné na implementáciu diferencovateľných verzii funkcií používaných v digitálnom spracovaní signálu. Delí na šesť modulov.

Prvým je modul *Core*, ktorý obsahuje diferencovateľné funkcie digitálneho spracovania signálu. Tieto funkcie sú následne využívané v signálových procesoroch, ktoré sú obsiahnuté vo zvyšných moduloch. Okrem toho modul *Core* obsahuje matematické operácie dôležité pri spracovávaní vstupných dát (napríklad funkciu, ktorá zabraňuje logaritmom záporných čísel, funkcie na výpočet logaritmov atď.), funkcie na prevody medzi jednotkami (napr. prevod amplitúdy z lineárnej mierky do logaritmickej atď.), škálovacie funkcie na úpravu dát a operácie so signálmi (konvolúcia, resampling) [14].

Druhým modulom základnej knižnice DDSP je *Processor*. V ňom je definovaný hlavný objektový typ *processor*, z ktorého dedia ostatné signálové procesory definované v moduloch *Synths* a *Effects*. Okrem toho je tam definovaná trieda *ProcessorGroup*, ktorá umožňuje združenie objektov typu *Processor* do skupín a triedu *Mix*, ktorá umožňuje prelínanie hlasitostí medzi signálmi [14].

Modul *Synths* definuje triedy zameranú na generovanie zvukového signálu. Vstupnými parametrami môžu byť výstupy neurónovej siete. V rámci riešenia problematiky reverzného inžinieringu mixáže tento modul nebudeme využívať, nakoľko sa naša problematika zaoberá mixážou zvukových signálov, nie ich tvorbou. Triedy syntezátorov definovaných v module *Synths* sú využité v iných aplikáciach spracovania zvukového signálu, napríklad vo výskume popísanom v [14].

Pre našu aplikáciu najdôležitejším modulom je modul *Effects*, ktorý implementuje diferencovateľné verzie efektov bežne používaných v digitálnom spracovaní zvukového signálu a ktoré budeme využívať pri zostavovaní reťazca spracovávajúceho vstupnú zvukovú stopu do výsledného mixu. Modul implementuje efekty *Reverb*, *Filter* a *ModDelay*, pričom v našom reťazci využívame efekty *Reverb* a *Filter*. Modul umožňuje implementáciu konvolučného dozvuku ako aj dozvuku, ktorého impulzná odozva je parametrizovaný exponenciálny útlm (anglicky *Exponential Decay*).

Knižnica ďalej obsahuje modul *Losses*, ktorý je knižnicou stratových funkcií aplikovateľných pri strojovom učení a modul *Spectral Ops*, ktorý obsahuje funkcie súvisiace s transformáciami signálu, napríklad FFT (rýchla Fourierova transformácia), či STFT (Krátkodobá Fourierova transformácia) [14].

2.2 Signálové procesory v DDSP

Základným princípom DDSP je použitie signálových procesorov používaných pri digitálnom spracovaní signálu v aplikáciach s využitím neurónových sietí. Z toho dôvodu sú procesory implementované tak, aby boli použiteľné ako vrstva neurónovej siete pomocou triedy *Processor* [14]. Táto implementácia umožňuje využitie v aplikáciach, v ktorých sú výstupy neurónovej siete použité ako parametre jednotlivých diferencovateľných signálových procesorov, napríklad za účelom prenosu *timbru* [14] (pozn. *timber* označuje subjektívnu kvalita zvuku súvisiacu s jeho spektrom [15]).

2.2.1 Trieda Processor

Trieda *Processor* je základným objektom DDSP, z ktorého dedia signálové procesory typu *Synths* a *Effects*. Trieda deklaruje metódy, ktoré sú implementované v jednotlivých signálových procesoroch. Týmito metódami sú `get_controls()`, `get_signal()` a `__call__()`.

Metóda `get_controls()` je implementovaná tak, aby upravila vstupné dáta (tenzory), ktoré sú obvykle výstupnými dátami neurónovej siete, pre účely spracovania signálu signálovým procesorom. Vstupné dáta signálového procesoru nemusia spĺňovať predpokladaný tvar, s ktorým procesor pracuje (najmä v prípade použitia výstupov neurónovej siete ako vstupných parametrov), takže signálový procesor nemusí byť schopný s týmito dátami relevantne pracovať. Metóda zabezpečí úpravu vstupných dát na parametre signálového procesoru a vráti slovník platných parametrov založených na zadaných vstupoch [16].

Metóda `get_signal()` implementuje samotné spracovanie signálu na základe parametrov predaných metóde. Najčastejšie sú týmito parametrami dáta, ktoré vráti metóda `get_controls()`, teda je zabezpečené, že sú valídnymi parametrami pre daný signálový procesor. Spracovanie signálu je definované individuálne v každom signálovom procesore, či už je z modulu *Synths* alebo *Effects*. Metóda vracia tensor audio dát, prípadne kontrolného signálu pre iný signálový procesor [16].

Implementácia metódy `__call__()` v jazyku Python umožňuje správanie sa inštancie objektu ako funkcie. V prípade *DDSP* signálových procesorov je táto metóda definovaná tak, aby zabezpečila úpravu priamu úpravu audio dát na základe vstupov. Metóda `get_controls()` sa v tomto prípade volá interne [16].

Trieda *Processor* dedí z triedy *Layer* knižnice *Keras*, zameranej na strojové učenie. V praxi to znamená, že všetky objekty a triedy dediace z triedy *Processor* je možné používať ako vrstvy modelu neurónovej siete. Triedy dediace z triedy *Processor* je vďaka tomu možné použiť v zmysle vrstvy neurónovej siete a pomocou metódy `add_weight` je možné vytvoriť trénovateľné premenné, váhy, ktoré sa aktualizujú v jednotlivých iteráciách počas trénovacieho cyklu modelu.

2.2.2 Efekty v DDSP

Z pohľadu reverzného inžinieringu mixáže s využitím knižnice *DDSP* je najdôležitejším modulom knižnice modul *DDSP Effects*, kde sú vytvorené triedy často používaných digitálnych zvukových efektov s využitím diferencovateľných funkcií. Modul obsahuje tri skupiny používaných audio efektov, a to efekty na úpravu dozvuku (*Reverb*), *filter* a triedu označenú ako *ModDelay*, ktorá umožňuje využitie oneskorovacích liniek s premenlivou dĺžkou [17].

V rámci projektu zameraného na reverzný inžiniering mixáže pomocou neurónovej siete budú využité dva efekty z modulu `ddsp.effects` a to *Reverb* a *Filter*. Za účelom vytvorenia reťazca signálových procesorov porovnateľného s mixážnym pultom bolo nutné implementovať dva efekty, ktoré modul `Effects` neobsahuje, a to efekty *Gain*, ktorý upravuje hlasitosť signálu a *Pan*, ktorý upravuje umiestnenie zvukovej stopy v stereofónnom obraze signálu.

Efekt Gain

Efekt *Gain* je pre účely projektu matematicky formulovaný ako:

$$Gain_{\text{out}}(x(n)) = G \times x(n)$$

kde $Gain_{\text{out}}(x(n))$ označuje výstupný signál efektu *Gain*, $x(n)$ vstupný signál efektu a G parameter efektu *Gain*, ktorý určuje hodnotu zosilnenia signálu.

Výpis 2.1: Implementácia efektu *Gain* v jazyku Python

```
class Gain(ddsp.processors.Processor):

  def __init__(self,scale_fn=None, name='Gain',
  trainable=False):

    super().__init__(name=name, trainable=trainable)
    self.scale_fn = scale_fn
    self.build()

  def build(self):
    if self.trainable:
      initializer = tf.keras.initializers.
      RandomUniform(minval=0.0, maxval=1.0)
      self._gain = self.add_weight(
      name = 'gain',
      shape = [1],
      dtype=tf.float32,
      initializer=initializer
      )
    self.built = True

  def get_controls(self, audio, gain):
    return {'audio': audio, 'gain': gain}

  def get_signal(self, audio, gain=None):
    if self.trainable:
      return audio * self._gain
    else:
      return audio * gain
```

V programe je definovaná trieda `Gain`, ktorá dedí z triedy `Processor`. Nakoľko vstupnými parametrami efektu nie sú výstupy neurónovej siete, v rámci metódy

`get_controls()` nebola implementovaná žiadna úprava vstupných dát a metóda vracia slovník vstupných parametrov. V rámci efektu je definovaná metóda `build`, ktorá je volaná pri vytvorení objektu triedy. V rámci tejto metódy je v prípade, že je vstupný argument `trainable` nastavený na hodnotu `True`, vytvorená váha pomocou metódy `add_weight`, s definovaným tvarom `1`, teda pole s jedným prvkom. Parameter `dtype` je dátový typ váhy (v našom prípade `tf.float32`), čo v praxi znamená, že hodnota bude desatinné číslo. Parameter `initializer` určuje počiatočnú hodnotu váhy. Na jej vytvorenie je použitý inicializátor z knižnice *Keras*, ktorý inicializuje váhu náhodnou hodnotou v rozsahu od $[0, 1]$.

Efekt Pan

Efekty typu *Panning* v stereofónií sú založené na princípe rozdielnej hlasitosti signálu do ľavého a pravého výstupného kanála. Implementácia efektu `Pan` v tejto bakalárskej práci vychádza z lineárneho priebehu efektu *Pan* (pôvodne z angličtiny *Linear Panning Law*) [18]. Matematicky sa dá vyjadriť nasledovnými rovnicami:

$$Pan_{\text{outL}}(x(n)) = \theta \times x(n)$$

$$Pan_{\text{outR}}(x(n)) = 1 - Pan_{\text{outL}}(x(n))$$

kde $Pan_{outL}(x(n))$ a $Pan_{outR}(x(n))$ označujú výstupný signál efektu do ľavého, resp. pravého kanála, θ je hodnota v rozsahu od 0 do 1 a určuje hlasitosť signálu v ľavom kanáli a tým zároveň pomer hlasitostí signálu medzi ľavým a pravým kanálom.

Pre použitie v rámci práce s `ddsp` je efekt implementovaný s využitím nelineárneho priebehu úrovne hlasitosti ľavého kanála použitím funkcie tanh [1], nakoľko spracovanie signálu efektom bude použité v zmysle aktivačnej funkcie vrstvy neurónovej siete. Matematicky je potom problém formulovaný:

$$Pan_{\text{outL}}(x(n)) = (0{,}5 + (0{,}5 \times \tanh(\Theta))) \times x(n)$$

$$Pan_{\text{outR}}(x(n)) = 1 - Pan_{\text{outL}}(x(n))$$

kde Θ je parameter ovplyvňujúci pomer medzi ľavým a pravým kanálom a je v rozsahu $(-\infty, \infty)$. Z formulácie vyplýva, že ak bude $\Theta = 0$, signál bude v stereofónnom obraze umiestnený presne v strede [1] .

Implementácia efektu v jazyku Python s využitím `ddsp` je uvedená na Výpise 2.2

Výpis 2.2: Implementácia efektu *Pan* v jazyku Python

```
class Pan(ddsp.processors.Processor):
  def __init__(self, scale_fn=None, name='Pan',
  trainable=False):

    super().__init__(name=name, trainable=trainable)
    self.scale_fn = scale_fn
    self.build()
  def build(self):
    if self.trainable:
    initializer = tf.keras.initializers.
    RandomUniform(minval=0.0, maxval=1.0)
    self._pan = self.add_weight(
      name = 'pan',
      shape = [1],
      dtype=tf.float32,
      initializer=initializer
    )
    self.built=True

  def get_controls(self, audio, pan):
    return {'audio': audio, 'pan': pan}
  def get_signal(self, audio, pan=None):
    if self.trainable:
      pan_l = (0.5 + (0.5*tf.math.tanh(self._pan)))
      pan_r = (1-pan_l)
      panned_signal_l = (pan_l * audio)
      panned_signal_r = (pan_r * audio)
      return [panned_signal_l, panned_signal_r]

    else:
      pan_l = (0.5 + (0.5*tf.math.tanh(pan)))
      pan_r = (1-pan_l)
      panned_signal_l = pan_l * audio
      panned_signal_r = pan_r * audio
      return [panned_signal_l, panned_signal_r]
```

Obdobne ako pri efekte `Gain` dedí trieda `Pan` zo základnej triedy `Processor`. V tomto prípade opäť nevyužívame implementáciu predspracovania vstupných para-

metrov metódou `get_controls()`, rovnako ako v prípade efektu `Gain` vracia metóda slovník vstupných parametrov.

V rámci triedy `Pan` je rovnako ako v triede `Gain` implementovaná metóda `build`, ktorá v prípade že parameter `trainable=True` vytvorí váhu, ktorá je optimalizovateľná v priebehu tréningu neurónovej siete. Váha `self._pan` je inicializovaná rovnakým spôsobom ako v prípade inicializácie váhy v rámci triedy `Gain`, nakoľko sa jedná o jednu číselnú hodnotu parametru *Pan*.

V metóde `get.signal` sú podmienkou `if...else...` definované prípady, kedy sa v prípade, že parameter `trainable=True`, vykoná výpočet *Panningu* s váhou vytvorenou v metóde `build`. V iných prípadoch je použitý vstupný parameter metódy `get.signal`, `pan`.

Efekt Reverb

Efekt `Reverb` je implementovaný ako signálový procesor v module knižnice `Effects`. Je popísaný ako konvolučný reverb, teda signálový procesor využívajúci signálovú operáciu *konvolúcia*. Matematicky sa spracovanie vstupného signálu procesorom dá popísať:

$$Reverb_{\text{out}}(x(n)) = \theta_{\text{Reverb}}(n) * x(n)$$

kde $\theta_{\text{Reverb}}(n)$ značí impulznú odozvu ako parameter *Reverbu*. Signálový procesor umožňuje zmiešanie procesovaného signálu (anglicky *wet*) s pôvodným, čistým signálom (anglicky *dry*). V takom prípade je problém formulovaný ako:

$$Reverb_{\text{out}}(x(n)) = \theta_{\text{Reverb}} * x(n) + x(n)$$

Efekt `Reverb` pracuje pri spracovaní signálu pomocou metódy `get_signal`, kde očakáva ako argumenty vstupné audio dáta a impulznú odozvu (označenú *ir*, z angličtiny *impulse response*). Na tie potom aplikuje konvolúciu zavolaním funkcie `ddsp.core.fft_convolve()` [14].

V prípade, že je parameter `trainable` nastavený na hodnotu `True`, je impulzná odozva `self._ir` implementovaná v zmysle váhy vrstvy neurónovej siete. Jej tvar je daný vstupným parametrom `reverb_length`, ktorý určuje dĺžku impzulznej odozvy vo vzorkoch [19]. Váha je inicializovaná inicializátorom implementovaným v knižnici *Tensorflow* s názvom `normal_initializer`, ktorý generuje tenzor určeného tvaru s normálnym rozdelením [20].

Výpis 2.3: Metóda `build` signálového procesoru `Reverb` [14]

```
    def build(self, unused_input_shape):
    """Initialize impulse response."""
    if self.trainable:
    initializer = tf.random_normal_initializer(mean=0,
    stddev=1e-6)
    self._ir = self.add_weight(
    name='ir',
    shape=[self._reverb_length],
    dtype=tf.float32,
    initializer=initializer)
    self.built = True
```

Knižnica okrem triedy `Reverb` implementuje triedu `ExpDecayReverb`, ktorá ako *impulznú odozvu* používa funkciu definovanú:

$$\theta_{\text{Reverb}} = gain \times \mathrm{e}^{-(2+\mathrm{e}^{decay})\cdot t} \times noise$$

kde *gain* a *decay* sú parametre nastavujúce amplitúdovú obálku priebehu šumu a *noise* označuje šum [14]. Šum je generovaný funkciou `random.uniform` z knižnice *TensorFlow*, ktorá v tomto prípade generuje pole dĺžky danej inicializovanou dĺžkou *Reverbu*, kde sú jednotlivé prvky generované náhodne s rovnomerným rozdelením pravdepodobnosti [21].

V jazyk Python je vytvorenie impulznej odozvy implementované metódou `_getir`, uvedenej na Výpise 2.4

Výpis 2.4: Metóda `_getir()` signálového procesoru `ExpDecayReverb` [14]

```
  def _get_ir(self, gain, decay):
    gain = self._scale_fn(gain)
    decay_exponent = 2.0 + tf.exp(decay)
    time = tf.linspace(0.0, 1.0, self._reverb_length)
           [tf.newaxis, :]
    noise = tf.random.uniform([1, self._reverb_length],
           minval=-1.0, maxval=1.0)
    ir = gain * tf.exp(-decay_exponent * time) * noise
    return ir
```

V prípade použitia efektu `ExpDecayReverb` sú ako váhy vrstvy použité dva parametre, a to parameter `self._gain` a `self._decay`. Optimalizované sú tak iba

dva parametre, z ktorých sa následne vypočíta impulzná odozva, nie celá impulzná odozva [19].

Efekt FIR Filter

Pojem *FIR filter* označuje digitálny číslicový filter s konečnou impulznou odozvou. Impulzná charakteristika takéhoto filtra má konečnú dĺžku a výstupný signál sa dá vyjadriť pomocou matematickej operácie konvolúcia:

$$y[n] = h(n) * x(n)$$

kde $h(n)$ je impulzná odozva *FIR filtra* a $x(n)$ je vstupný signál.

V knižnici `ddsp` je *FIR filter* implementovaný pomocou objektu `ddsp.FIRFilter`. Metóda `get_signal` filtra, ktorá bude využívaná v rámci mixáže očakáva dva vstupné argumenty, pričom parametrom `audio` je predaný vstupný signál filtru a parametrom `magnitudes` je predaná frekvenčná prenosová krivka. V rámci metódy `get_signal` je zavolaná funkcia z modulu `core` s názvom `frequency_filter`. Táto funkcia ako prvé volá funkciu z toho istého modulu s názvom `frequency_impulse_response` vytvorí impulznú odozvu z frekvenčnej prenosovej funkcie predanej prostredníctvom `magnitudes` [14].

Následne funkcia `frequency_filter` zavolá funckiu `fft_convolve`, ktorá vykoná konvolúciu obdobne ako je to v prípade efektu *Reverb*.

Implementácia efektu EQ

Nakoľko samotný efekt *FIR filter* neimplementuje metódu, ktorá by vytvorila a inicializovala váhu v prípade, že bol parameter `trainable` nastavený na hodnotu `True`, v rámci použitia tohto efektu ako vrstvy neurónovej siete sme implementovali triedu `EQ`, ktorá dedí z triedy `Processor`. Trieda implementuje metódu `build`, v ktorej je definovaná váha vrstvy pomocou metódy `add._weight`. V tomto prípade je váhou frekvenčná charakteristika filtru, definovaná tenzorom s tvarom `(1, 1025)`. Z toho vyplýva, že optimalizovanou váhou je 1025 vzorková frekvenčná prenosová funkcia. Inicializovaná je pomocou inicializátoru z knižnice *Keras* s označením *RandomUniform*, ktorý inicializuje váhu hodnotami danými rovnomerným rozdelením.

Samotné spracovanie signálu prebieha pomocou objektu triedy `FIRFilter`, ktorý je vytvorený lokálne pri inicializácií objektu triedy `EQ`. V metóde `get_signal` triedy `EQ` sa volá metóda `get_signal` objektu `self.filter`, ktorá vykoná spracovanie signálu efektom *FIR filter* popísané vyššie.

Výpis 2.5: Implementácia metódy `build` procesoru `EQ`

```
    def build(self):
    if self.trainable:
    initializer = tf.keras.initializers.
    RandomUniform(minval=0.0, maxval=1.0)
    self._magnitudes = self.add_weight(
    name = 'magnitudes',
    shape = (1, 1025),
    dtype=tf.float32,
    initializer=initializer
    )
    self.built=True
```

V prípade, že je parameter `trainable` nastavený na hodnotu `True`, ako frekvenčná prenosová funkcia je použitá váha vytvorená v metóde `build`, v iných prípadoch je použitá charakteristika špecifikovaná vo vstupných parametroch.

Výpis 2.6: Implementácia metódy `get_signal` procesoru `EQ`

```
  def get_signal(self, audio, magnitudes=None):
    if self.trainable:
      return self.filter.get_signal(tf.reshape(audio,
      (1, tf.size(audio))), self._magnitudes)
    else:
      return self.filter.get_signal(audio, magnitudes)
```

Samotná frekvenčná odozva je modelovaná na základe desaťpásmového oktávového ekvalizéru [22]. Ten môže byť charakterizovaný sadou desiatich parametrov $\theta_{EQ_{gains}}$ [1], pomocou ktorých sú nastavené zosilnenia na frekvenciách zodpovedajúcich stredným frekvenciám filtrov používaných v oktávovom ekvalizéri (31, 63, 125, 250, 500, 1000, 2000, 4000, 8000 a 16000 Hz) [22].

Parametre $\theta_{EQ_{\text{gains}}}$ v našej implementácií určujú zosilnenie frekvencií v pásme širokom:

$$B = f_{\max} - f_{\min}$$

kde $f_{\min} = \frac{f_c}{\sqrt{2}}$ a $f_{\max} = f_c \cdot \sqrt{2}$, kde f_c je stredný kmitočet daný strednými kmitočtami oktávového ekvalizéru [22]. V takto určených pásmach je frekvenčná charakteristika modelovaná ako ideálna pásmová priepusť, teda všetky frekvencie spadajúce do pásma B sú zosilnené zosilnením určeným parametrami $\theta_{EQ_{gains}}$ na

danej strednej frekvencií. V praxi je pri transformácií frekvenčnej odozvy na impulznú odozvu použité váhovacie okno (v prípade efektu *FIRFilter* z *DDSP Hannovo oknom* s dĺžkou rovnou dĺžke impulznej odozvy [19]) zabezpečujúce kauzalitu filtru a konečnosť impulznej odozvy. Zároveň dochádza k zníženiu strmosti prechodu z priepustného do nepriepustného pásma [23].

Frekvenčná odozva je implementovaná funkciou `design_bandpass_response`, ktorá je uvedená v Prílohe A.1.

3 Mixáž zvuku pomocou DDSP

3.1 Všeobecné princípy mixáže zvuku

Mixáž zvuku je popísaná ako proces zmiešania jednotlivých prvkov zvukového diela do výsledného mixu. Zvukový technik úpravou jednotlivých stôp pomocou rozličných signálových procesorov a efektov vytvára výsledný, najčastejšie stereofónny, mix[1]. Cieľom mixáže zvukového diela je vytvorenie subjektívne kvalitne znejúcej nahrávky, ktorá spĺňa isté objektívne parametre súvisiace so žánrovým zaradením nahrávky, preto je spôsob zmiešania stôp je čiastočne vedou a čiastočne umením [24].

Mixáž zvuku má viacero funkcií. Prvou, najdôležitejšou je vyváženie stôp získaných počas nahrávacieho procesu, a to v zmysle frekvenčného a hlasitostného vyváženia. Okrem toho je jej cieľom aj úprava technických abnormalít, ktoré vznikli počas procesu nahrávania. V neposlednom rade je mixáž zvuku kreatívny proces, ktorý výrazným spôsobom ovplyvňuje vznik hudobného diela vo forme nahrávky [24].

Zámerom mixáže by malo byť dosiahnutie vyváženej výslednej stopy, a to vyváženej v oblasti hlasitosti, frekvenčného spektra a stereofónneho obrazu. Postupy, ktorými je možné vyváženosti dosiahnuť je možné zhrnúť do piatich základných kategórií: *Úprava frekvenčného spektra*, *Korekcia hlasitosti*, *Úprava stereofónneho obrazu*, *Dozvuk* a *Úprava dynamiky*. V rámci mixáže použitej v tejto semestrálnej práci sú využité úpravy v prvých štyroch kategóriach, nakoľko knižnica *DDSP* neimplementuje procesory využívané na úpravu dynamiky signálu.

3.2 Signálový reťazec spracovania zvukovej stopy

Signálový reťazec mixáže pomocou `ddsp` sa skladá z efektov popísaných v 2.2.2. Matematicky si ho formulujme ako:

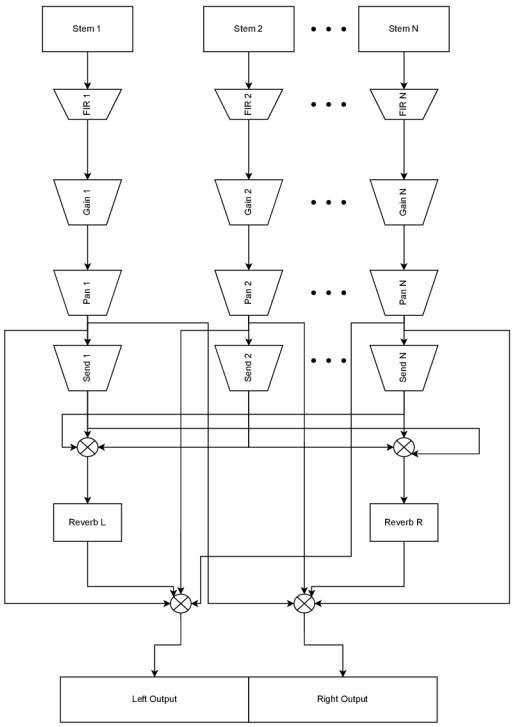
$$y_{\mathrm{L}} = \sum_{i=1}^{N} Pan_{\mathrm{L}}(Gain(FIR(x(n)_i))) + \sum_{i=1}^{N} Reverb_{\mathrm{L}}(Send(Pan_{\mathrm{L}}(Gain(FIR(x(n)_i)))))$$

$$y_{\mathrm{R}} = \sum_{i=1}^{N} Pan_{\mathrm{L}}(Gain(FIR(x(n)_i))) + \sum_{i=1}^{N} Reverb_{\mathrm{R}}(Send(Pan_{\mathrm{R}}(Gain(FIR(x(n)_i)))))$$

kde $\hat{y}_L$ a $\hat{y}_R$ sú ľavý, resp. pravý výstupný kanál po mixáži. *Send* označuje špeciálny prípad implementácie efektu *Gain*, ktorý určuje, s akým zosilnením je signál privádzaný do efektu *Reverb*.

Takto popísaný reťazec pre N spracovávaných vstupných stôp sa dá zobraziť diagramom na 3.1. Zvolené poradie efektov pre ciele našej mixáže zodpovedá zobrazeniu.

Obr. 3.1: Diagram signálového reťazca použitého pri mixáži



Pri mixáži sa najskôr signál vstupnej stopy (na diagrame označený ako *Stem*) filtruje pomocou *FIR filtra* s frekvenčnou odozvou modelovanou podľa 10-pásmového oktávového ekvalizéru popísaného v kapitole 2.2.2, následne je jeho signálová úroveň upravená efektom *Gain* a *Pan*. Signál jednotlivých kanálov (ľavého a pravého) za efektom *Pan* je upravený pomosou *Sendu*, popísaného vyššie. Signál z jednotlivých stôp za efektom *Send* sa sčítava a vstupuje do efektov *Reverb*, určených pre ľavý a pravý kanál. Výstupný signál s *Reverbov* sa sčíta so súčtom signálov ľavého a pravého kanála odoberaných pred efektom *Send*. Týmto spôsobom je docielená simulácia N-kanálového mixážneho pultu s efektom *Reverb* použitým na stereofónnej zbernici, pričom signály sú odoberané *post-fader* [25].

V rámci praktického prevedenia mixáže pomocou knižnice *DDSP* bola implementovaná trieda `Channel_strip`. V konštruktore triedy sú inicializované inštancie tried *Gain*, *Pan* a *EQ*, ktoré sú následne využité na postupné spracovanie vyššie popísaným reťazcom.

Vstupný argument `trainable` je nastavený na hodnotu `False`, nakoľko chceme využiť užívateľom zadané hodnoty parametrov, nie interne inicializované váhy signálových procesorov.

Samotné spracovanie je definované v metóde *Processing*, ktorá preberá ako argu-

Výpis 3.1: Implementácia triedy `Channel_strip`

```
class Channel_strip():
def __init__(self):
self._Gain = Gain(trainable=False)
self._Pan = Pan(trainable=False)
self._EQ = EQ(trainable=False)

def processing(self, path, gain, pan, eq_gains):
data = wav_load(path)
mags = self.design_bandpass_response(eq_gains)
panned_L = self._Pan.get_signal(self._Gain.get_signal
   (self._EQ.get_signal(data, mags), gain), pan)[0]
panned_R = self._Pan.get_signal(self._Gain.get_signal
   (self._EQ.get_signal(data, mags), gain), pan)[1]
return [panned_L, panned_R]
```

menty cestu umiestnenia zdrojového zvukového súboru (označenú ako `Path`) a ďalej argumenty `gain` (predanie parametru zosilnenia efektu *Gain*), `pan` (parameter efektu *Pan*) a `eq_gains` (pole definujúce zosilnenia jednotlivých stredných kmitočtov, podľa ktorých je modelovaná kmitočtová odozva, popísaná v 2.2.2. Následne je do premenných `panned_L` a `panned_R` uložený ľavý a pravý spracovaný signál, po postupnom spracovaní signálovými procesormi *EQ*, *Gain* a *Pan*. Signál je upravený najskôr v kmitočtovej oblasti, následne je upravená jeho hlasitosť a nakoniec rozloženie v priestore.

3.2.1 Použitie efektu Reverb pri mixáži

Implementácia efektu *Reverb* pri mixáži je v tejto práci založená na princípe dvoch signálových procesorov z knižnice *DDSP* typu `ExpDecayReverb`, popísaného v kapitole 2.2.2. Jednotlivé signálové procesory sú priradené ľavému a pravému kanálu a ich vstupným signálom je mix jednotlivých stôp spracovaných signálovým reťazcom pozostávajúcim z predošlých spomínaných efektov s pridaným spracovaním pomocou signálového procesoru označeného ako *Send*. *Send* pracuje na princípe efektu *Gain*, pričom určuje úroveň, s akou bude signál danej stopy prítomný na vstupe efektu *Reverb*.

Signály za efektom *Send* sú zmiešané jednotlivo pre ľavý a pravý kanál a následne sú spracované efektami *Reverb*. V implementácií je argument `add_dry` signálových procesorov *Reverb* nastavený na hodnotu `False`, čo zabezpečuje, že na výstupe je

prítomný iba signál po spracovaní efektom *Reverb*, bez primiešania vstupného signálu.

Po spracovaní je signál z oboch *Reverbov* primiešaný k ľavému a pravému kanálu stereofónneho mixu signálov odoberaných pred *Sendom*.

Výpis 3.2: Implementácia spracovania signálu *Reverbom* pri mixáži šesťkanálového mixu

```
1  add = ddsp.processors.Add()
2  Reverb = ddsp.effects.ExpDecayReverb(reverb_length
       =44100, add_dry=False)
3
4  reverb_gain = -1.5
5  reverb_decay = 0.5
6
7  ReverbedL = Reverb(add.get_signal(Verb_1L, add.
       get_signal(Verb_2L, add.get_signal(Verb_3L,    add.
       get_signal(Verb_4L, add.get_signal(Verb_5L, Verb_6L)
       )))), reverb_gain, reverb_decay)
8
9  ReverbedR = Reverb(add.get_signal(Verb_1R, add.
       get_signal(Verb_2R, add.get_signal(Verb_3R, add.
       get_signal(Verb_4R, add.get_signal(Verb_5R, Verb_6R)
       )))), reverb_gain, reverb_decay)
```

Procesor `Add` z knižnice *DDSP* slúži na zmiešanie jednotlivých signálov. Parametre `reverb_gain` a `reverb_decay` boli pri mixáži nastavené sluchom. Na základe nastavených parametrov je vytvorená impulzná odozva a následnou konvolúciou vstupného signálu s takto namodelovanou impulznou odozvou dostávame výstupný signál.

Impulzné odozvy oboch použitých *Reverbov* vystupujú v zmysle váh vrstiev neurónovej siete, teda sú hľadanými parametrami pri spätnej remixáži pomocou neurónovej siete.

3.3 Proces mixáže

V rámci bakalárskej práce bolo vytvorených päť referenčných vzoriek mixov s dĺžkou pohybujúcou sa od 10 do 30 sekúnd. Tieto mixy boli následne použité ako cieľové dáta pri učení neurónovej siete, pričom snahou bolo, aby sa mix vytvorený neurónovou sieťou čo najviac približoval cieľovému mixu.

Mixáž bola uskutočnená pomocou spracovania jednotlivých vstupných stôp reťazcom spracovania signálu popísaným v 3.2, pričom parametre boli nastavené poslúchom podľa vnútorných estetických kritérií zvukového majstra (pozn. autora práce).

Pri mixáži sa pracovalo s upravenými vstupnými stopami piatich skladieb, ktorých viacstopové záznamy sú dostupné voľne na akademické účely. Pochádzajú z knižnice viacstopových nahrávok s názvom *The Mixing Secrets* [18]. Z nahrávok boli urobené výstrižky o dĺžke v rozmedzí 10 až 30, následne zmiešané do stereofónneho mixu.

Nakoľko samotný proces mixáže musel prebiehať prostredníctvom programovania v jazyku Python, bolo potrebné vytvoriť efektívny spôsob približujúci sa skutočnej mixáži zvukových stôp. Z toho dôvodu bol program na vytváranie jednotlivých mixov implementovaný vo webovom rozhraní *Google Collab*, ktoré umožňuje spúšťanie kódu napísaného v jazyku Python vo webovom prehliadači, a implementuje základné grafické rozhranie, čo umožnilo vytvoriť základné grafické prvky na ovládanie jednotlivých parametrov.

Rozhranie okrem toho umožňuje spúšťanie kódu v jazyku Python po takzvaných *bunkách*, ktoré umožňujú spustenie len vybranej časti kódu. Pri zmene jedného parametru tak nie je nutné opätovne spúšťať celý kód, je dostačujúce spustiť íba modifikovanú časť.

3.3.1 Predspracovanie zvukových súborov

Jednotlivé používané súbory boli uložené vo formáte *Waveform Audio File Format* (skrátene *WAV*) so vzorkovacím kmitočtom 44100 Hz a jednotlivé zvukové vzorky boli reprezentované 32bitovou premennou typu *float* (dátový typ umožňujúci ukladanie desatinných čísel).

Predspracovanie dát v jazykuPython prebiehalo prostredníctvom implementovanej funkcie `wav_load`, ktorá využíva funkciu z knižnice *SciPy* nazvanú `wavfile.read`. Pomocou tejto funkcie sa do poľa `data` načítajú jednotlivé vzorky zvukového signálu, pričom každá vzorka je reprezentovaná 32bitovou premennou typu *float*.

Následne funkcia upraví pole vstupných hodnôt na tvar požadovaný pri spracovaní pomocou signálových procesorov z knižnice *DDSP*. Tvar vstupného tenzoru obsahujúceho vstupné zvukové dáta je podľa dokumentácie knižnice [19] 2-D tenzor, pričom v jednotlivých stĺpcoch sú uložené samotné vzorky zvukových dát, a počet riadkov určuje počet takzvaných *batchov* [19] (z angličtiny *dávka*), teda skupín dát, ktoré sú použité na trénovanie v rámci jedného trénovacieho cyklu.

V prípade implementácie použitej pri mixáži bude výsledný tvar dát po potrebnej úprave 2-D pole s tvarom $(1 \times n_{\text{samples}})$, kde n_{samples} označuje počet načítaných zvukových vzoriek.

Táto implementácia pre ďalšie spracovanie predpokladá, že počet zvukových vzoriek načítaných do jednotlivých stôp je rovnaký, teda prvá načítaná stopa má n vzoriek, predpokladá sa, že ostatné načítané stopy budú mať tiež n vzoriek. V opačnom prípade vznikne chyba pri výslednom sčítaní stôp do výsledného mixu pri použití efektu `Add` v poslednom kroku mixáže, nakoľko spomínaný signálový procesor vykonáva, ako vyplýva z kódu, sčítanie vzorky po vzorke, teda je potrebné, aby vstupné tenzory mali rovnaký tvar [19]. V prípade, že počet vzoriek jednotlivých stôp nesedí, je potrebné upraviť stopy skrátením, prípadne pridaním nulových vzoriek. V prípade, že by vstupným súborom bol viackanálový zvukový súbor, funkcia načíta iba vzorky prvého kanála

Implementovaná funkcia ako argument preberá cestu umiestnenia súboru (označenú ako `path`) a je zobrazená na Výpise 3.3. Na úpravu dát je použitá metóda `reshape`, ktorá umožňuje zmeniť tvar poľa, pričom zachová dáta uchované v poli. Prvým argumentom je počet riadkov výsledného poľa a druhým je počet stĺpcov výsledného poľa, pričom pri použití argumentu -1 je počet stĺpcov zanechaný z pôvodného poľa [26].

Výpis 3.3: Implementácia načítania a predspracovania zvukových dát

```
1  def wav_load(path):
2    samples, data = wavfile.read(path)
3    if len(data.shape) == 2:
4      data = data[:, 0]
5
6    data = data.reshape(1, -1)
7    return data
```

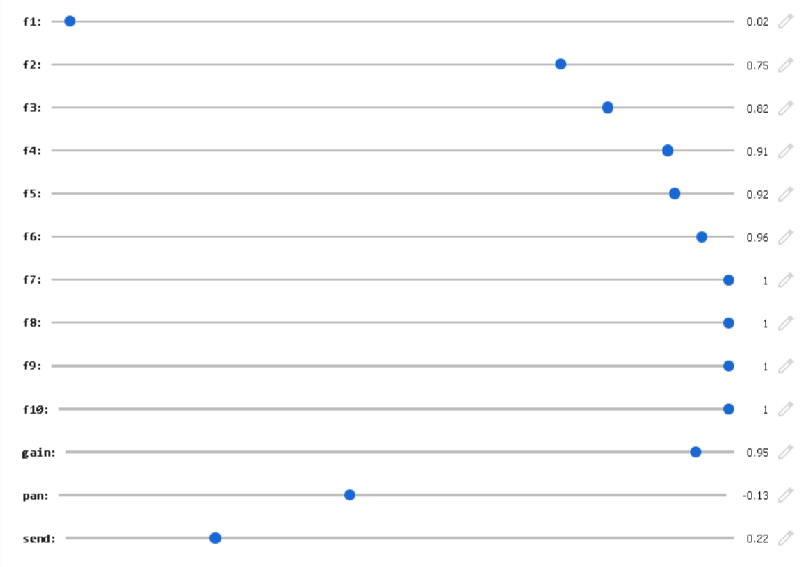
3.3.2 Rozhranie určené na mixáž zvuku

Za účelom zjednodušenia procesu mixáže bolo prostredníctvom webového rozhrania *Google Colab* vytvorené rozhranie, ktoré umožňuje ovládať jednotlivé parametre použité pri mixáži pomocou grafických prvkov. *Google Colab* umožňuje jednotlivým premenným definovať ovládací prvok, ktorého zmenou sa mení hodnota samotnej premennej [27]. Tento ovládací prvok môže mať tvar ako textového poľa, tak posuvného ovládacieho prvku. Ukážka rozhrania je zobrazená na obrázku 3.2

Takto vytvorené grafické rozhranie umožňuje nastavovanie parametrov jednotlivých signálových procesorov podobným spôsobom, akým sú parametre prestavované na mixážnom pulte, prípadne v programoch na spracovanie zvuku.

Parametre označené ako `f1` až `f10` označujú zosilnenia na jednotlivých frekvenciách oktávového ekvalizéru, popísaného v kapitole 2.2.2. Parameter `gain` nastavuje

Obr. 3.2: Ukážka nastavovateľných prvkov grafického rozhrania vytvoreného pomocou *Google Colab*



zosilnenie efektu *Gain* priradeného danej stope, parameter `pan` nastavuje umiestnenie vo výslednom stereofónnom obraze a `send` nastavuje úroveň, s akou sa signál z danej stopy posiela do efektu *Reverb*. Takáto sada grafických prvkov je vytvorená pre každú stopu. Vytvorený súbor vo formáte .ipynb je súčasťou elektronickej prílohy.

3.3.3 Export dát do výsledného stereofónneho zvukového súboru

Po spracovaní jednotlivých stôp pomocou triedy `Channel_strip`, efektu *Reverb* a sčítaní upravených stôp pomocou procesoru `Add` sú na výstupe reťazca dve polia obsahujúce dáta ľavého a pravého kanála, ktoré je nutné uložiť vo formáte prehrateľného zvukového súboru.

Za týmto účelom bola implementovaná funkcia `stereo_audio_to_wav`. Funkcia preberá ako argumenty dáta ľavého kanála (`left_channel`), dáta pravého kanála (`right_channel`), vzorkovací kmitočet výsledného zvukového súboru (`sample_rate`) a meno súboru, prípadne cestu, kam má byť súbor uložený (`filename`). Pomocou funkcie z knižnice *NumPy* s názvom `vstack`, ktorá zlúči polia do viacrozmerného poľa s tvarom $(n_{\text{arrays}} \times n_{\text{samples}})$, kde n_{arrays} je počet zlúčených polí a n_{samples} počet vzoriek v poli, pričom všetky zlúčené polia musia obsahovať rovnaký počet prvkov [28].

Uloženie do súboru prebieha pomocou funkcie z knižnice *SciPy* s označením `wavfile.write()`. Aby bolo možné do súboru *WAV* zapísať dáta z ľavého a pra-

vého kanála, je nutné ako argument funkcie predať pole s rozmermi ($n_{\text{samples}} \times n_{\text{arrays}}$), kde počet riadkov zodpovedá počtu vzoriek zapisovaných dát a počet stĺpcov zodpovedá počtu zapisovaných kanálov. Na tento účel slúži funkcia `ndarray.T`, ktorá transponuje viacrozmerné pole v jazyku Python.

Takto formátované dáta sú zapisované do zvukového súboru so vzorkovacím kmitočtom určeným argumentom `sample_rate` do súboru určeného argumentom `filename`. Popísaná implementácia je v jazyku Python zobrazená na Výpise 3.4. Výsledný audio súbor má vzorkovací kmitočet 44100 Hz a vzorky sú reprezentované 32bitovou hodnotou typu *float*.

Výpis 3.4: Implementácia exportu zvukových dát

```
1  def stereo_audio_to_wav(left_channel, right_channel,
       sample_rate, filename):
2    audio = np.vstack((left_channel, right_channel)).T
3    wavfile.write(filename, sample_rate, audio)
```

3.3.4 Vytvorené mixy

V rámci bakalárskej práce bolo vytvorených 5 žánrovo odlišných mixov. Zdrojom viacstopových nahrávok bola voľne dostupná knižnica *The Mixing Secrets* [18]. V rámci skladieb boli z jednotlivých stôp vytvorené skrátené úseky v dĺžke od 10 do 40 sekúnd, ktoré boli následne zmiešané do výsledných stereofónnych mixov. Počet mixovaných stôp sa pohyboval v rozmedzí od 5 do 8.

Prvou zmiešavanou skladbou bola skladba autora *Bena Flowersa* s názvom *Ecstasy*, ktorá by sa dala žánrovo zaradiť do oblasti pomalšieho *electro popu*, teda kombinácie elektronickej a populárnej hudby [18]. V rámci mixáže tejto skladby bolo zmiešaných päť vstupných stôp, pričom výrazným prvkom bola vokálna stopa. Parametre efektov použitých pri mixáži sú uvedené v Tabuľke B.1 v Prílohe B.1.

Druhá zmiešaná skladba má názov *I'm Alright* od kapely *Angels in Amplifiers* [18]. Jedná sa o *rock popovú* skladbu s akustickou aj elektrickou gitarou a mužským vokálom. Zmiešavaných bolo 7 vstupných stôp o dĺžke 21 sekúnd. Parametre nastavenia efektov sú uvedené v Tabuľke B.2 v Prílohe B.2.

Tretí mix vznikol mixážou elektronickej skladby žánru *techno* s názvom *Ubiquitous* autora *Alberta Kadera* [18]. Skladba sa vyznačuje výrazným partom bicích so zameraním na basový bubom, vo frekvenčnej oblasti hlboko položenou basovou linkou a výraznými syntezátormi. Mixovaných bolo 6 vstupných stôp s dĺžkou 18 sekúnd. Parametre použité pri mixáži sú uvedené v Tabuľke B.3 v Prílohe B.3.

Preposledný mix bol vytvorený mixážou nahrávky žaňrovo spadajúcej do klasickej hudby. Jedná sa o skladbu *Johanna Sebastiana Bacha*, kantátu s názvom *Je-*

sus bleibet meine Freude. Mixovaná inštrumentácia je napísaná pre hoboj, čembalo a sláčikové kvarteto a vokál. Nástroje sú nahrané na 7 mikrofónov pričom dva tvoria stereofónny pár v systéme *ORTF* [29], dva tvoria techniku *Mid-side* [29] a tri boli použité na kontaktné snímanie sláčikového kvarteta a hlasu [18]. V rámci tejto práce bolo v mixáži využitých 5 mikrofónov, pričom bol vynechaný pár tvoriaci *Mid-side* techniku. Zo skladby bol vybraný 21 sekundový inštrumentálny úsek. Paramtere použité pri mixáži sú uvedené v Tabuľke B.4.

Posledným mixom zmiešaným v rámci tejto bakalárskej práce je skladba zaraditeľná do žánru *country pop* [18]. Autorkou skladby je *Anna Blanton* a skladba sa volá *Rachel*. Mix bol vytvorený zo 7 vstupných stôp, pričom sa jedná o spev a akustické nástroje v zložení violončelo, kontrabas, akustická gitara, ukulele, perkusívne nástroje a viola. Pre účely práce boli stopy upravené na dĺžku 39 sekúnd a parametre použité pri mixáži sú zhrnuté v Tabuľke B.5.

Okrem popísaných mixov v rámci práce bola zmiešaná druhá sada mixov, ktoré boli zmiešané s rovnakými parametrami bez zaradenia efektu *Reverb*.

4 Reverzný inžiniering mixáže pomocou neurónovej siete

Reverzný inžiniering mixáže je možné popísať ako hľadanie takých parametrov jednotlivých efektov aplikovaných pri spracovaní stôp do výsledného zvukového mixu, aby sa výsledný mix čo najviac približoval mixu referenčnému. Majme signál $y(n)$ reprezentujúci referenčný mix, vytvorený pomocou signálových procesorov s nastavenými parametrami Θ, ktorého parametre chceme určiť a signál $\hat{y}(n)$ vytvorený signálovým reťazcom charakterizovaný súborom parametrov $\hat{\Theta}$. Cieľom je určenie takého súboru parametrov $\hat{\Theta}$, aby bol rozdiel medzi $y(n)$ a $\hat{y}(n)$, reprezentovaný zvolenou stratovou funkciou, čo najmenší [1]. Signál $\hat{y}(n)$ je v prípade reverzného ínžinieringu mixáže pomocou neurónovej siete reprezentovaný výstupom neurónovej siete so špeciálnou štruktúrou pozostávajúcou z diferencovateľných signálových procesorov z knižnice *DDSP*, teda súbor parametrov $\hat{\Theta}$ symbolizuje váhy jednotlivých vrstiev neurónovej siete.

Predpokladaný je teda signálový reťazec popísaný v kapitole 3.2, ktorého výstupný stereofónny dvojkanálový signál je reprezentovaný signálmi $y_{\mathrm{L}}(n)$ pre signál ľavého kanálu a $y_{\mathrm{R}}(n)$ pre signál pravého kanálu. Je predpoklad, že tieto signály boli zmiešané pomocou efektov z knižnice *DDSP*, popísaných v kapitole 2.2.2, s použitím sady parametrov jednotlivých efektov označenej Θ.

Ďalej majme neurónovú sieť so špeciálnou štruktúrou skladajúcou sa z efektov knižnice *DDSP* zoradených rovnakým spôsobom ako signálový reťazec popísaný v kapitole 3.2. Jednotlivé signálové procesory reprezentujú vrstvy neurónovej siete, kde parametre efektov $\hat{\Theta}$ vystupujú v zmysle prestaviteľných váh siete. Výstupy takejto siete $\hat{y_{\mathrm{L}}}(n)$ a $\hat{y_{\mathrm{R}}}(n)$ reprezentujú stereofónny mix skladby spracovaný použitím parametrov $\hat{\Theta}$. Cieľom je minimalizovať $\|y_{\mathrm{L}}(n) - \hat{y_{\mathrm{L}}}(n)\|$ a zároveň minimalizovať $\|y_{\mathrm{R}}(n) - \hat{y_{\mathrm{R}}}(n)\|$, kde $\|\cdot\|$ reprezentuje použitú stratovú funkciu.

Minimalizácia chyby reprezentovanej stratovou funkciou prebieha aktualizáciou váh v procese spätného šírenia pomocou optimalizátora *Adam*, popísaného v kapitole 1.3.2. Ako stratová funkcia bola použitá *stredná kvadratická odchýľka* popísaná v kapitole 1.3.1.

4.1 Štruktúra siete

Štruktúra použitej siete je navrhnutá tak, aby v doprednom kroku aplikovala spracovanie signálu reálne využívané v procese spracovania zvukového signálu a procese mixáže [1]. Použitými vrstvami sú diferencovateľné verzie signálových procesorov

implementované pomocou knižnice *DDSP*, pričom jednotlivé efekty sú medzi sebou prepojené v rámci spracovania jednotlivých vstupov.

Vstupmi neurónovej siete sú predspracované zvukové dáta spracované pomocou algoritmu 3.3.1. Predspracovanie dát zabezpečuje, aby boli dáta neurónovej sieti predané v definovanom tvare vhodnom pre spracovanie jednotlivými vrstvami siete.

Jeden vstup siete je spracovávaný práve jednou kombináciou signálových procesorov *Gain*, *Pan*, *EQ* a *Send*. Topológia siete je identická s reťazcom spracovania signálu použitým pri mixáži, ktorá je uvedená na diagrame 3.1, pričom parametrami efektov použitých pri spracovaní signálu sú váhy jednotlivých vrstiev siete.

Implementovaná neurónová sieť má dva výstupy, ktoré zodpovedajú ľavému a pravému kanálu stereofónneho mixu po spracovaní signálovými procesormi s parametrami zodpovedajúcim váham neurónovej siete.

Dopredný krok neurónovej siete je v tomto prípade definovaný rovnako ako je definovaná mixáž pomocou signálového reťazca poísaného v kapitole 3.2, pričom pre každý vstup sú inicializované všetky signálové procesory, bez ohľadu na to, či boli skutočne použité pri mixáži referenčného mixu. Predpokladáme, že pri mixáži referenčného signálu boli použité a nastavené všetky signálové procesory z popisovaného reťazca spracovania signálu.

V rámci bakalárskej práce bol okrem modelu neurónovej siete s popísanou štruktúrov navrhnutý a implementovaný model so zredukovanou štruktúrou, bez zaradenia efektu *Reverb*. Takýto model by sa matematicky dal popísať vzťahmi definujúcimi výstupné signály ľavého a pravého kanálu ako:

$$\hat{y}_{\mathrm{L}} = \sum_{i=1}^{N} Pan_{\mathrm{L}}(Gain(FIR(x(n)_i)))$$

$$\hat{y}_{\mathrm{R}} = \sum_{i=1}^{N} Pan_{\mathrm{R}}(Gain(FIR(x(n)_i)))$$

V práci je skúmaný ako plne implementovaný, tak zjednodušený model.

4.2 Inicializácia modelu v jazyku Python

Praktická implementácia vyššie popísanej štruktúry je založená na vytvorení modelu neurónovej siete prostredníctvom knižníc *Tensorflow* a API (*Application programming interface* [30]) *Keras*.

Model bol vytvorený dedením z triedy `keras.Model` [31]. V rámci práce boli implementované dva modely, jeden bez zaradenia efektu *Reverb*, ktorý je označený ako `Test_model_filter`, a jeden s plnou štruktúrou so zaradením všetkých efektov, označený ako `Test_model_reverb`. Modely sú implementované ako samostatné

triedy, kde v konštruktore `__init__` je definovaná štruktúra vrstiev siete a metóde `call` je definovaný dopredný krok prechodu dát neurónovou sieťou.

Konštruktor preberá jeden vstupný parameter, a tým je definovaný počet vstupných kanálov. Podľa tohto parametru sa následne inicializuje daný počet objektov signálových procesorov tak, aby zodpovedali poísanej štruktúre spracovania signálu neurónovou sieťou. Všetky signálové procesory sú vytvorené s argumentom `trainable=True`, čím je zabezpečená inicializácia parametrov signálového procesoru ako učiteľnej premennej, váhy.

V prípade modelu `Test_model_reverb` sú inicializované dva signálové procesory `ddsp.effect.Reverb)`, s parametrom `trainable` nastaveným na hodnotu `True` a parametrom `reverb_length` na `44100`, čo v praxi znamená, že učiteľnou váhou je v tomto prípade impulzná odozva s dĺžkou 44100 vzorkov [19].

Počiatočné hodnoty jednotlivých váh sú inicializované pomocou inicializátorov implementovaných v *Kerase* [32]. Počiatočná hodnota parametru efektu *Gain* je náhodné číslo v rozsahu od 0 do 1, rovnakým spôsobom je inicializovaná aj hodnota *Sendu*. Hodnota parametru *Pan* je inicializovaná náhodným číslom v rozsahu od -1 do 1, pričom záporné hodnoty parametru zodpovedajú umiestneniu v stereofónnom obraze smerom do prava, kladné smerom doľava.

Váha vrstvy tvorenej signálovým procesorom *EQ* je inicializovaná 2-D tenzorom s tvarom $(1, 1025)$, ktorého hodnoty sú vytvorené pomocou `RandomUniform()` inicializátora [32]. Hodnotami poľa sú čísla v rozsahu od 0 do 1.

Obdobne sú inicializované váhy vrstiev tvorených signálovými procesormi *Reverb*. V tomto prípade je inicializovaný 2-D tenzor s tvarom $1, 44100$, pričom hodnoty sú inicializované pomocou inicializátora `RandomNormal`, ktorý naplní pole náhodnými vzorkami s normálovým rozdelením [32]. Priemer definujúci použité normálne rozdelenie je rovný 0 a smerodajná odchýľka je rovná 10^{-6}. I-tá vzorka takto inicializovaného poľa by sa dala matematicky vyjadriť:

$$T_{1,i} \sim \mathcal{N}(0, 10^{-6})$$

Zdrojový kód tried `Test_model_filter` a `Test_model_reverb` sú dostupné v Prílohe A.2. Definície implementácií signálových procesorov boli v rámci práce uložené v súbore `model_layers.py`.

4.3 Implementácia dopredného kroku v jazyku Python

Pri implementácií modelu prostredníctvom dedenia z triedy `keras.Model` musí byť dopredný krok definovaný v metóde `call` [31]. Metóda má dva parametre, `self`

je referenciou na konkrétnu inštanciu triedy a `inputs` označuje vstupný argument, ktorým metóda preberá vstupné dáta neurónovej siete.

V ďalších riadkoch metódy je definovaný samotný výpočet výstupného signálu dopredného kroku siete. Jedná sa o postupné spracovanie vstupných dát, predspracovaných postupom popísaným v kapitole 3.3.1, vrstvami neurónovej siete reprezentovaných popisovanými signálovými procesormi. Vrstvy siete zodpovedajú reťazcu spracovania signálu použitého pri mixáži.

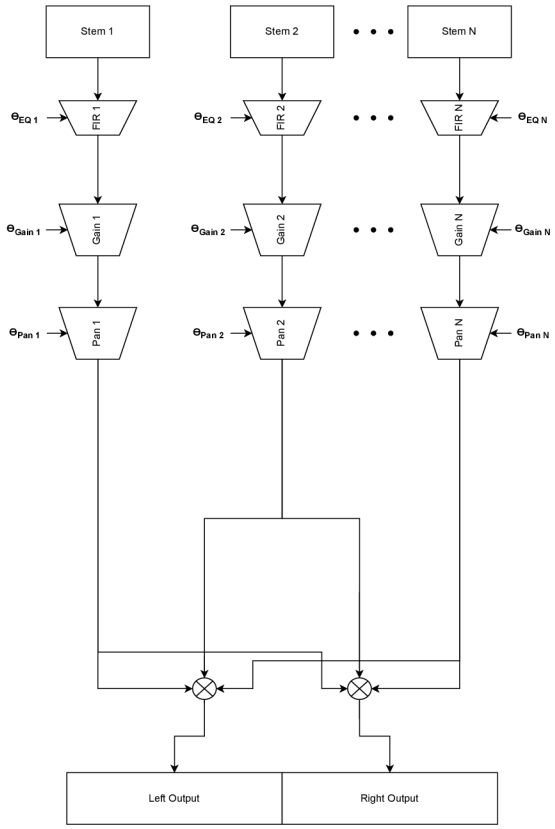
V prípade modelu s označením `Test_model_filter` sa jedná o spracovanie vstupných stôp signálovým reťazcom pozostávajúcim z efektov *EQ*, *Gain* a *Pan*, čomu v programe postupné volanie metód *get_signal* jednotlivých signálových procesorov v poradí *EQ*, *Gain* a nakoniec *Pan*. V programe je viditeľné, že najskôr sa popísaným spôsobom vypočíta výstupný signál zodpovedajúci spracovaniu prvého vstupu neurónovej siete. Signál pre ľavý kanál sa uloží do premennej `added_L`, signál pre pravý kanál sa ukladá do premennej `added_R`. Nakoľko signálový procesor *Pan* vracia pole dvoch hodnôt, pričom prvá hodnota určuje signál ľavého kanála a druhá hodnota určuje signál pravého kanála, do premennej `added_L` je uložená prvá hodnota vráteného poľa (určená indexom [0]) a do premennej `added_R` druhá hodnota vráteného poľa (určená indexom [1]).

Následne sa využitím podmienkového cyklu `for` postupne pričítajú spracované signály zvyšných vstupných kanálov, pričom sa súčet ukladá do už definovaných premenných `added_L` a `added_R`. Za účelom sčítania signálov je využitá metóda `get_signal` efektu implementovaného v knižici *DDSP* s názvom `Add`, ktorá sčíta vstupné dáta prvok po prvku [19].

Po sčítaní dát metóda vracia pole, v ktorom sú uložené premenné `added_L` a `added_R`, ktoré sú zároveň výstupmi neurónovej siete. Štruktúra modelu je zobrazená na diagrame 4.1, kde parametre Θ vyjadrujú váhy neurónovej siete.

V prípade modelu `Test_model_reverb` je dopredný krok založený na podobnom princípe. Metóda `call` preberie v rámci argumentu `inputs` vstupy neurónovej siete, teda predspracované vstupné stopy (predspracovanie popísané v kapitole 3.3.1). Do premenných `panned_L` a `panned_R` sa uloží signál spracovaný reťazcom spracovania signálu, pričom posledným signálovým procesorom spracovávajúcim signál je efekt *Pan*. Do premenných `verbed_L` a `verbed_R` sa uloží signál určený pre ďalšie spracovanie efektom *Reverb*. Takýto signál je spracovaný signálovým reťazcom zloženým z rovnakých signálových procesorov ako ten, ktorý je uložený do premenných `panned_L` a `panned_R`, pričom je navyše spracovaný signálovým proscesorom *Send*. Ten určuje, s akou úrovňou bude signál spracovanej vstupnej stopy spracovávaný efektom *Reverb* 3.2. Premenné `added_L` a `added_R` sú v tomto bode naplnené hodnotami uloženými v premenných `panned_L`, resp. `panned_R`.

Rovnako ako v prípade modelu `Test_model_filter` je spracovanie jednotlivých



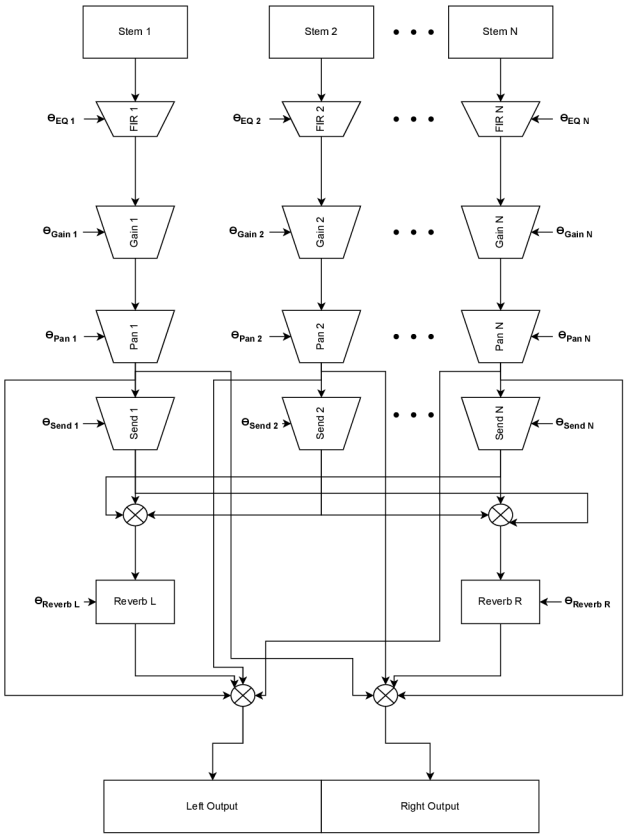
Obr. 4.1: Diagram zobrazujúci spracovanie vstupného signálu modelom bez spracovania efektom *Reverb*

vstupov siete vrstvami a ich následné sčítanie vyriešené pomocou cyklu `for`. V cykle sa najskôr vypočíta výstupný signál po spracovaní i-tého vstupu signálovým reťazcom *EQ*, *Gain* a *Pan*. Výstupné signály procesoru *Pan* sú uložené do premenných `panned_L` a `panned_R` a tie sú pripočítané k premenným `added_L` a `added_R`. Následne sa k premennej `verbed_L` pripočíta hodnota premennej `panned_L`, spracovaná signálovým procesorom *Send* a k premennej `verbed_R` je pripočítaná rovnakým spôsobom spracovaná hodnota premennej `panned_R`.

Hodnoty uložené v premenných `verbed_L` a `verbed_R` slúžia ako vstupné argumenty metód `get_signal` signálových procesorov `_reverb_L` a `_reverb_R`, pričom druhými vstupnými argumentami oboch metód sú impulzné odozvy efektov `_reverb_L`, resp. `_reverb_R`, vystupujúce v zmysle váh neurónovej siete. Tie sú uložené v premenných `_reverb_L._ir` a `_reverb_R._ir`. Po spracovaní sú dáta znovu uložené do premenných `verbed_L` a `verbed_R`.

Na záver sa hodnoty uložené v premennej `added_L` sčítajú s hodnotami premennej `verbed_L`, resp. hodnoty v premennej `added_R` s hodnotami `verbed_R` a uložia

sa do premenných `added_L` a `added_R`, ktoré funkcia `call` vracia ako návratové hodnoty. Tie sú výstupmi modelu neurónovej siete. Principiálna štruktúra popísaného modelu je zobrazená na diagrame 4.2, kde parametre Θ predstavujú jednotlivé váhy neurónovej siete.



Obr. 4.2: Diagram zobrazujúci spracovanie vstupného signálu modelom so spracovaním pomocou efektu *Reverb*

4.4 Hľadanie parametrov použitých pri mixáži pomocou neurónovej siete

Po skončení dopredného kroku neurónovej siete sú v premenných `added_L` a `added_R` dáta zodpovedajúce výstupom ľavého a pravého kanála stererofónného mixu po zmiešaní popísaným reťazcom efektov s parametrami Θ. Tieto dáta je možné označiť ako predikciu neurónovej siete [2].

Nech je takto vektor zmiešaného výstupu ľavého kanála označený $\hat{y}_{\mathrm{L}}$ a výstupu pravého kanála označený $\hat{y}_{\mathrm{R}}$. Cieľom je nájsť taký súbor parametrov Θ, aby sa

$\hat{y}_\mathrm{L}$ a $\hat{y}_\mathrm{R}$ čo najviac približovali vektorom ľavého a pravého kanála y_L a y_R cieľového mixu, teda mixu, ktorého parametre chceme odhadnúť. V prípade, že chybu pravého kanála označíme $\|y_\mathrm{R}(n) - \hat{y}_\mathrm{R}(n)\|$ a chybu ľavého kanála $\|y_\mathrm{L}(n) - \hat{y}_\mathrm{L}(n)\|$, je možné predpokladať, že použitím parametrov Θ pri mixáži sa budú signály $\hat{y}_\mathrm{L}$ a $\hat{y}_\mathrm{R}$ líšiť od cieľových signálov y_L a y_R práve o túto chybu.

Ak by bola chyba v ideálnom prípade minimalizovaná na 0, bolo by možné povedať, že pri použití parametrov Θ sa $\hat{y}_\mathrm{L} = y_\mathrm{L}$ a $\hat{y}_\mathrm{R} = y_\mathrm{R}$, teda použitím parametrov Θ je možné dosiahnuť rovnakého výsledného mixu ako je mix s hľadanými parametrami.

Cieľom je teda nastaviť súbor váh neurónovej siete Θ vystupujúcich v zmysle parametrov efektov použitých pri mixáži tak, aby bola minimalizovaná chyba medzi hodnotami ľavých a pravých kanálov predikovaného a cieľového mixu. Hľadanie parametrov cieľového mixu je preto možné definovať ako učenie neurónovej siete s váhami Θ.

Váhy sú prestavované v procese spätného šírenia, popísaného v kapitole 1.3.2. Ako stratová funkcia je v implementáciách oboch modelov v rámci tejto práce použitá *stredná kvadratická odchýlka*, popísaná v kapitole 1.3.1. Chybu ľavého a by bolo možné matematicky popísať:

$$E_\mathrm{L} = \frac{1}{N}\sum_{i=1}^{N}(y_{\mathrm{L}_i} - \hat{y}_{\mathrm{L}_i})^2$$

kde y_{L_i} označuje i-tý prvok vektora y_L a $\hat{y}_{\mathrm{L}_i}$ označuje i-tý prvok vektora $\hat{y}_\mathrm{L}$.

Rovnakým spôsobom je možné vyjadriť chybu pravého kanála:

$$E_\mathrm{R} = \frac{1}{N}\sum_{i=1}^{N}(y_{\mathrm{R}_i} - \hat{y}_{\mathrm{R}_i})^2$$

Celková chyba je vyjadrená ako:

$$E = E_\mathrm{L} + E_\mathrm{R}$$

Použitým optimalizačným algoritmom je *Adam*, popísaný v kapitole 1.3.2. Po implementačnej stránke je pri trénovaní siete využitá možnosť API *Keras*, ktoré automatickú diferenciáciu a zostup gradientu s využitím rôznych algoritmov pomocou metódy `model.fit()`. Pred samotným trénovacím cyklom, implementovaným použitím `model.fit()`, je nutné model nakonfigurovať, na čo slúži metóda `model.compile()`, ktorá zabezpečí správne nakonfigurovanie optimalizátora a stratovej funkcie [31].

V rámci bakalárskej práce prebiehal tréningový proces pre každú mixovanú skladbu, pričom hľadanie parametrov prebieha na oboch verziách mixov skladby, ako so zaradeným efektom *Reverb*, tak bez neho. Pri hľadaní parametrov dvoch mixov

jednej skladby sú trénované dva modely, `Test_model_filter` a `Test_model_reverb`. Tento proces bol opakovaný pre každú z piatich skladieb, popísaných v kapitole 3.3.4.

Model `Test_model_filter` bol trénovaný pomocou optimalizátora *Adam* s počiatočnou rýchlosťou učenia (*learning rate*, vysvetlené v kapitole 1.3.2 rovnou hodnote $0,01$. S týmito parametrami je vykonaných 500 iterácií, pričom je implementované skoré zastavenie učenia (anglicky *early stopping*). Skoré zastavenie v praxi znamená, že v prípade, že sa chyba nezníži v danom určenom počte iterácií po sebe, tréningový proces sa zastaví. Po prvom cykle učenia s vyššie poísanými parametrami sa *learning rate* zníži na 10^{-3} a následne po 200 iteráciach na 10^{-4}, pričom sa vykoná posledných 200 iterácií v prípade, že nenastane podmienka skorého zastavenia, ktorá je v celom procese nastavená na 10 iterácií.

Praktická implementácia v jazyku Python s využitím metód `model.fit()` a `model.compile()` je uvedená na Výpise 4.1.

Výpis 4.1: Implementácia tréningového procesu pre model bez zaradenia efektu Reverb

```
1  model1.compile(optimizer=tf.optimizers.Adam(
       learning_rate=0.01), loss='mse')
2  callback = tf.keras.callbacks.EarlyStopping(monitor=
       'loss',patience=10) #nastavenie skoreho
       zastavenia, pricom proces je zastaveny v pripade,
        ze sa strata neznizi v 10 iteraciach po sebe
3  history = model1.fit(x=[Ch_1, Ch_2, Ch_3, Ch_4, Ch_5,
       Ch_6, Ch_7], y=[target_L, target_R],    epochs
       =500, batch_size=1, callbacks=[callback])
4  model1.compile(optimizer=tf.optimizers.Adam(
       learning_rate=0.001), loss='mse')
5  history = model1.fit(x=[Ch_1, Ch_2, Ch_3, Ch_4, Ch_5,
       Ch_6, Ch_7], y=[target_L, target_R],  epochs=200,
       batch_size=1, callbacks=[callback])
6  model1.compile(optimizer=tf.optimizers.Adam(
       learning_rate=0.0001), loss='mse')
7  history = model1.fit(x=[Ch_1, Ch_2, Ch_3, Ch_4, Ch_5,
       Ch_6, Ch_7], y=[target_L, target_R], epochs=200,
       batch_size=1, callbacks=[callback])
```

Prístup v prípade učenia model `Test_model_reverb` je takmer identický s rozdielom, že model je trénovaný na väčšom počte iterácií, z dôvodu zvýšenia náročnosti výpočtu problému v súvislosti s pridaním váh dvoch efektov typu *Reverb*, ktorých

váhami sú impulzné odozvy definované každá 44100 vzorkami. Praktická implementácia v jazyku Python je zobrazená na Výpise 4.2. V rámci úpravy bol zvýšený počet iterácií s rýchlosťou učenia 10^{-4}.

Výpis 4.2: Implementácia tréningového procesu pre model so zaradením efektu Reverb

```
1    model2.compile(optimizer=tf.optimizers.Adam(
        learning_rate=0.01), loss='mse')
2    callback = tf.keras.callbacks.EarlyStopping(monitor='
        loss',patience=20)
3    history = model2.fit(x=[Ch_1, Ch_2, Ch_3, Ch_4, Ch_5,
         Ch_6, Ch_7], y=[target_L, target_R], epochs=500,
        batch_size=1, callbacks=[callback])
4    model2.compile(optimizer=tf.optimizers.Adam(
        learning_rate=0.001), loss='mse')
5    history = model2.fit(x=[Ch_1, Ch_2, Ch_3, Ch_4, Ch_5,
         Ch_6, Ch_7], y=[target_L, target_R], epochs=1000,
         batch_size=1, callbacks=[callback])
6    model2.compile(optimizer=tf.optimizers.Adam(
        learning_rate=0.0001), loss='mse')
7    history = model2.fit(x=[Ch_1, Ch_2, Ch_3, Ch_4, Ch_5,
         Ch_6, Ch_7], y=[target_L, target_R], epochs=200,
        batch_size=1, callbacks=[callback])
```

V rámci procesu učenia popísaných modelov je počet optimalizovaných váh závislý na počte vstupných stôp, pričom pre každú stopu sú definované váhy Θ_{gain}, ktorá je vyjadrená práve jednou hodnotou, Θ_{pan}, takisto vyjadrená jednou hodnotou, Θ_{EQ} vyjadrená poľom 1025 hodnôt. V prípade modelu s implementovaným spracovaním efektom *Reverb* sa ku každej stope pridá váha Θ_{Send}, ktorá je vyjadrená jednou číselnou hodnotou a váhy Θ_{ReverbL} a Θ_{ReverbR} vyjadrujúce impulzné odozvy efektov *Reverb*, pričom každá je reprezentovaná poľom 44100 hodnôt. S pridaním váh, teda optimalizovaných parametrov, vzrastá výpočetná náročnosť problému, nakoľko algoritmus pri prestavovaní váh prestavuje väčšie množstvo parametrov.

Tréningový proces bol vykonaný na 10 sekundových úsekoch vybraných z jednotlivých skladieb. Použité rýchlosti učenia a počty iterácií boli určené experimentálne tak, aby sieť dosiahla posluchovo prijateľného výsledku, pričom presnosť výsledkov dosiahnutých pri tomto nastavení parametrov je popísaná v kapitole 5.1.

5 Porovnanie výsledkov

Výsledkom tejto práce je sada mixov vytvorených spätnou remixážou skladieb popísaných v kapitole 3.3.4, pričom ako parametre efektov boli použité váhy neurónovej siete po dokončení tréningového procesu popísaného v kapitole 4.4. Takto vzniknuté remixáže sú porovnané s cieľovými mixmi, ktoré boli použité ako cieľové hodnoty pri učení neurónovej siete.

Porovnanie remixáží a cieľových mixov bolo vykonané ako objektívne, tak subjektívne (posluchovým testom). V rámci objektívneho porovnávania boli remixáže porovnávané s cieľovými mixami v troch oblastiach a to v rámci hlasitosti, stereofónneho obrazu a spektrálnej informácie, pričom výsledky sú uvedené v kapitole 5.2. Subjektívne boli remixáže a cieľové mixy porovnané posluchovým testom formou dotazníku, popísaným v kapitole 5.3.

Okrem porovnania výsledných remixáží je výsledkom práce porovnanie samotných parametrov odhadnutých pomocou modelov s parametrami použitými pri mixáži cieľových mixov. Porovnanie je popísané v kapitole 5.1.

5.1 Porovnanie parametrov použitých pri mixe s parametrami určenými neurónovou sieťou

Porovnanie parametrov určených neurónovou sieťou bolo založené na určení chyby medzi určeným parametrov (váhou) a parametrom skutočne použitým pri mixáži skladby. V prípade váh založených na skalárnej hodnote je chyba určená ako absolútna chyba [33] medzi určenou hodnotou a hodnotou určenou pri mixáži:

$$\Delta\Theta = \Theta_{\text{odhad}} - \Theta_{\text{reálna}}$$

pričom Θ_{odhad} označuje hodnotu určenú neurónovou sieťou a $\Theta_{\text{reálna}}$ označuje hodnotu parametru použitú pri mixáži. Týmto spôsobom boli vypočítané chyby parametrov jednotlivých efektov *Gain*, *Pan* a *Send*.

V prípade určenia presnosti predikcie neurónovej siete pri parametroch efektu *EQ*, nakoľko sa jedná o porovnanie dvoch vektorov, bolo zvolené porovnávanie pomocou odstupu signálu od šumu (*SNR* signal-to-noise ratio) podľa vzťahu:

$$SNR_{\text{EQ}} = 20 \cdot \log\left(\frac{\|\boldsymbol{\Theta}_{\text{cieľ}}\|_2}{\|\boldsymbol{\Theta}_{\text{cieľ}} - \boldsymbol{\Theta}_{\text{odhad}}\|_2}\right)$$

kde $\boldsymbol{\Theta}_{\text{odhad}}$ označuje vektor frekvenčnej odozvy ekvalizéru určený neurónovou sieťou, $\boldsymbol{\Theta}_{\text{target}}$, označuje vektor frekvenčnej odozvy použitý pri mixáži cieľového mixu a $\|\cdot\|_2$ označuje Euklidovskú normu. Hodnota SNR je v decibeloch (dB), teda čím vyššia je hodnota, tým presnejší je odhad neurónovej siete.

V prípade analýzy presnosti určenia impulznej odozvy efektu *Reverb* bol tiež zvolený odstup signál-šum, pričom vypočítaný bol podľa vzťahu:

$$SNR_{\text{Reverb}} = 20 \cdot \log\left(\frac{\|\boldsymbol{\Theta}_{\text{cieľ}}\|_2}{\|\boldsymbol{\Theta}_{\text{cieľ}} - \boldsymbol{\Theta}_{\text{odhad}}\|_2}\right)$$

kde v tomto prípade $\boldsymbol{\Theta}_{\text{odhad}}$ označuje vektor predikcie impulznej odozvy efektu *Reverb* a $\boldsymbol{\Theta}_{\text{target}}$ označuje vektor impulznej odozvy použitej pri mixáži cieľového mixu.

Analyzované boli vždy dve sady parametrov určených sieťou a parametrov cieľového mixu, pričom jedna sada zodpovedala modelu `Test_model_filter` a druhá modelu `Test_model_reverb`.

Tabuľky s porovnaním sú uvedené v prílohe C. Výsledky porovnávania parametrov ukazujú, že vo veľa prípadoch sú parametre určené neurónovou sieťou výrazne odlišné od parametrov použitých pri mixáži cieľových mixov. Najpresnejšie určené boli parametre efektu *Pan* pri remixáži pomocou modelu `Test_model_filter`, pričom presnosť určenia tohto parametru je najprecíznejšia pri remixáži skladby *Ecstacy*.

Výrazné rozdiely v určených parametroch nemusia znamenať, že výsledná remixáž s použitím parametrov určených neurónovou sieťou nebude znamenať podobný výsledok, ako mixáž s pôvodným nastavením parametrov pri mixáži cieľového mixu. Výsledný mix vzniká kombináciou spracovania prostredníctvom jednotlivých efektov, pričom k rovnakému či podobnému výsledku je možné dôjsť rozličnými kombináciami nastavení parametrov jednotlivých efektov.

V konečnom dôsledku sú tak vnímaná hlasitosť či *timber* v nahrávke dané kombináciou parametrov použitých pri mixáži, pričom dve rôzne kombinácie môžu dosiahnuť rovnaký výsledok.

Práve z tohto dôvodu sú modely vyhodnocované nielen z pohľadu presnosti určenia parametrov použitých pri mixáži, ale aj na základe podobnosti cieľového mixu s remixážou vytvorenou pomocou parametrov určených modelmi, pričom remixáže boli s cieľovým mixom porovnávané objektívne aj subjektívne.

Kódy v jazyku Python použité pri porovnávaní sú uvedené v elektronickej prílohe v programe *evaluation.py*.

5.2 Objektívne porovnanie remixáží vytvorených parametrami určenými neurónovou sieťou s cieľovými mixami

V rámci práce boli objektívne porovnávané remixáže vytvorené pomocou neurónovej siete s ručnými mixážami popísanými v kapitole 3.3.4 na vybraných 10 sekundových

úsekoch. Všetky remixáže boli porovnávané s cieľovým mixom na úseku, na ktorom boli trénované, pričom skladby *Ecstasy* a *Rachel* boli následne remixované aj na inom 10 sekundovom úseku pomocou parametrov určených neurónovou sieťou.

Porovnania boli robené v oblasti hlasitosti, stereofónneho obrazu a spektra. V oblasti hlasitosti bolo porovnanie urobené na základe hlasitosti vypočítanej podľa normy ITU-R BS.1770 [34], ktorej výpočet je implementovaný v knižnici napísanej v jazyku Python s názvom *Pyloudnorm* [35]. Knižnica umožňuje jednoducho vypočítať hlasitosť zvukového súboru v jednotkách LUFS, vyplývajúcich z ITU-R BS.1770. Ukážka kódu implementujúceho výpočet hlasitosti je zobrazená na Výpise 5.1.

Výpis 5.1: Implementácia výpočtu hlasitosti pomocou knižnice Pyloudnorm

```
1    def compute_loudness(datas):
2      meter = pyln.Meter(44100)
3      return meter.integrated_loudness(datas)
```

Funkcia preberie zvukové dáta. Výpočet hlasitosti prebieha pomocou objektu `Meter`, ktorý preberá ako vstupný argument vzorkovací kmitočet, ktorý je v našom prípade pevne nastavený na 44100, nakoľko pracujeme s audio vzorkovaným týmto vzorkovacím kmitočtom. Následne funkcia vráti hodnotu integrovanej hlasitosti v jednotkách LUFS vypočítaných pomocou metódy `integrated_loudness` objektu `meter`.

Porovnanie v oblasti stereofónneho obrazu spočívalo v dvoch komparáciach, pričom prvá bola založená na porovnávaní integrovanej hlasitosti ľavého a pravého kanála remixáže a cieľového mixu, pričom druhá spočívala vo výpočte odstupu signálu od šumu, ktorý bol použitý už pri porovnávaní parametrov efektov v kapitole 5.1. Pre signál v ľavom kanále by sa dal matematicky zapísať:

$$SNR_{\mathrm{L}} = 20 \cdot \log\left(\frac{\|\mathbf{y}_{\mathrm{cieľ_L}}\|_2}{\|\mathbf{y}_{\mathrm{cieľ_L}} - \mathbf{y}_{\mathrm{odhad_L}}\|_2}\right)$$

kde $\mathbf{y}_{\mathrm{cieľ_L}}$ je signál ľavého kanála cieľového mixu, $\mathbf{y}_{\mathrm{odhad_L}}$ signál ľavého kanála remixáže.

Obdobne je možné napísať vzťah aj pre pravý kanál:

$$SNR_{\mathrm{R}} = 20 \cdot \log\left(\frac{\|\mathbf{y}_{\mathrm{cieľ_R}}\|_2}{\|\mathbf{y}_{\mathrm{cieľ_R}} - \mathbf{y}_{\mathrm{odhad_R}}\|_2}\right)$$

.

Pre vizuálne porovnanie boli vytvorené vektorskopy [36] ako jednotlivých cieľových mixov, tak remixáží.

Za účelom porovnania spektier bolo využité porovnanie spektrogramov ľavého a pravého kanála s využitím obdobného princípu založeného na odstupe signál-šum, pričom spektrogramy boli vytvorené z remixáže aj cieľového mixu pomocou knižnice jazyka Python *librosa* [37], zameranej na spracovanie zvukového signálu. Veľkosť použitého okna pri tvorbe spektrogramu bola 2048, pričom použité okno bolo *Hannovo okno*. Program na výpočet a zobrazenie spektrogramu je zobrazený na Výpise 5.2.

Výpis 5.2: Implementácia výpočtu a vykreslenia spektrogramu pre ľavý kanál pomocou knižnice librosa

```
1     def create_spect(inputs, sr=44100):
2       n_fft=2048
3       hop_length=512
4       ft_t=lib.stft(y=inputs, n_fft=n_fft, window=sp.
           signal.windows.hann, hop_length=hop_length)
5       D = np.abs(ft_t)
6       db = lib.amplitude_to_db(D, ref=np.max)
7       return db
8     lib.display.specshow(create_spect(data[:, 0]), sr
         =44100, y_axis='log', x_axis='time')
```

Princíp porovnania pomocou odstupu signálu od šumu bol aplikovaný aj na celý zvukový stereofónny signál, na základe čoho je možné porovnať presnosť remixáže pomocou modelu.

Tabuľky s výsledkami objektívneho porovnania Prílohe D. Vektorskopy a spektrogramy sú vo formáte .jpg priložené v elektronickej prílohe.

Z objektívneho porovnania vyplýva rozdiel presnosti remixáže pri zaradení efektu *Reverb* do reťazca spracovania signálu, pričom model implementujúci spracovanie signálu efektom *Reverb* dosahuje nižšej presnosti remixáže. Tento rozdiel môže byť spôsobený vyššou výpočetnou náročnosťou problému a komplexnosťou problému (pridanie váh definovaných poľom o počte 44100 prvkov, ako aj nižším počtom iterácií počas trénovacieho procesu. Trénovací cyklus bol obmedzený výpočetným výkonom zariadenia, na ktorom boli oba modely trénované, a to primárne z časových dôvodov.

Model so zaradeným efektom *Reverb* vykazuje horšie vlastnosti vo všetkých porovnávaných oblastiach, pričom najvýraznejšie sa rozdiel prejavuje v porovnaniach vykonaným na základe *SNR*.

5.3 Subjektívne porovnanie posluchovým testom

V rámci subjektívneho porovnania bol vytvorený dotazník založený na princípe podobnom testovaniu *ABX* [38], ktoré je založené na porovnávaní referenčnej zvukovej stopy s dvoma vzorkami, označnenými A a B. Jedna vzorka je zhodná s referenciou, pričom cieľom je určiť, ktorá so vzoriek A alebo B to je. Test sa používa pri hodnotení rozlíšiteľnosti rôznych zvukových vzoriek ľudským sluchom.

Posluchový test použitý v rámci tejto práce nie je možné za *ABX* test považovať, nakoľko nespĺňa viacero podmienok nutných pri vykonaní *ABX* testu, ako napríklad opakovaný posluch tých istých vzoriek, pričom po každé je správna odpoveď iná. Taktiež nebolo možné realizovať test v rovnakých podmienkach pre všetkých respondentov a nebolo možné zabezpečiť náhodné rozloženie otázok. Podmienky nebolo možné splniť z časových a materiálnych dôvodov. Subjektívny test je teda nutné považovať len za demonštratívny.

Použitý posluchový test vychádza z princípu porovnávania dvoch rozličných vzoriek, označených A a B s referenčnou vzorkou, pričom jedna z možností je zhodná s referenciou. Dotazník sa skladal z 10 otázok, pričom v každej otázke bol ako referencia použitý cieľový mix ktorý bol použitý na tréning neurónovej siete. Porovnávanými možnosťami boli zvuková stopa identická s referenciou a remixáž vytvorená modelom neurónovej siete. V rámci dotazníku boli použité ako remixáže vytvorené modelom `Test_model_filter`, tak modelom `Test_model_reverb`.

Úlohou respondenta bolo vybrať jednu z možností a následne v podotázke vybrať, na základe čoho druhú možnosť vyradil. Takáto modifikácia umožnila získať informáciu, v akej oblasti je rozdiel medzi referenčným mixom a remixážou najvýraznejší. V prípade, že sa respondent nevie rozhodnúť, mal otázku vynechať a v podotázke označiť odpoveď „Neviem určiť“. V prípade že svoju odpoveď vybral náhodne, mal označiť možnosť „Náhodný výber“.

V rámci prvej otázky boli porovnávané mixy skladby *Ecstasy*, pričom neurónová sieť bola trénovaná na porovnávanom úseku. Použitým modelom na vytvorenie remixáže bol v tomto prípade `Test_model_filter`. Správna odpoveď bola odpoveď A. Výsledky sú zhrnuté v tabuľke 5.3.

Počet správnych odpovedí je v tomto prípade nižší ako počet odpovedí nesprávnych v kombinácií s odpoveďou „Neviem posúdiť“, z čoho vyplýva náročná rozlíšiteľnosť medzi jednotlivými nahrávkami.

V druhej otázke sa jedná o mixáž skladby *Jesu bleibet meine Freude*, pričom bola mixáž aj remix boli vytvorené so zaradením efektu *Reverb*, použitím modelu **Test_model_reverb**. Správnou odpoveďou bola možnosť B. Výsledky sú zhrnuté v tabuľke 5.3.

Na základe odpovedí môžeme usúdiť, že v tomto prípade bolo určenie správnej

Tab. 5.1: Odpovede na otázku 1

Otázka 1	Počet odpovedí
A	9
B	4
Náhodný výber	0
Neviem posúdiť	6
Hlasitosť jednotlivých nástrojov	5
Hlasitosť celej zvukovej ukážky	2
Rozloženie nástrojov v priestore	0
Rozdiely vo frekvenčnom spektre	8
Rozdielny dozvuk	1

Tab. 5.2: Odpovede na otázku 2

Otázka 2	Počet odpovedí
A	3
B	13
Náhodný výber	0
Neviem posúdiť	3
Hlasitosť jednotlivých nástrojov	7
Hlasitosť celej zvukovej ukážky	3
Rozloženie nástrojov v priestore	8
Rozdiely vo frekvenčnom spektre	7
Rozdielny dozvuk	2

nahrávky jednoduchšie ako pri otázke 1, a to najmä na základe rozdielov v stereofónnom obraze a frekvenčnom spektre. Napriek tomu sa vyskytla časť odpovedí, ktoré neboli správne, prípadne respondenti neboli schopní určiť, ktorá nahrávka je identická s referenčnou. Výsledok naznačuje, že rozdiel medzi nahrávkami môže byť počuteľný, ale nie výrazný.

V rámci tretej otázky bola posudzovaná mixáž skladby *Rachel*, pričom bola použitá remixáž pomocou modelu `Test_model_filter`. Bol vybraný úsek, na ktorom bol model učený a výsledky sú zhrnuté v tabuľke 5.3. Správnou odpoveďou bola odpoveď A.

V tomto prípade na odpovediach pozorujeme signifikantnú časť nesprávnych odpovedí a odpovedí „Neviem posúdiť“, čo naznačuje náročnosť určenia správnej odpovede.

V rámci štvrtej otázky bola porovnávaná opäť skladba *Ecstasy*, tentokrát druhý úsek, ktorý bol remixovaný na základe parametrov určených neurónovou sieťou,

Tab. 5.3: Odpovede na otázku 3

Otázka 3	Počet odpovedí
A	11
B	3
Náhodný výber	0
Neviem posúdiť	5
Hlasitosť jednotlivých nástrojov	8
Hlasitosť celej zvukovej ukážky	0
Rozloženie nástrojov v priestore	3
Rozdiely vo frekvenčnom spektre	9
Rozdielny dozvuk	1

ktorá bola trénovaná na inom úseku skladby. Bol použitý model `Test_model_filter` a účelom je porovnanie rozlíšiteľnosti remixáže v prípade, že je sieť učená na porovnávanom úseku a v prípade, že je remixovaná iná časť skladby. Výsledky sú zhrnuté v tabuľke 5.4.

Tab. 5.4: Odpovede na otázku 4

Otázka 4	Počet odpovedí
A	10
B	2
Náhodný výber	0
Neviem posúdiť	7
Hlasitosť jednotlivých nástrojov	5
Hlasitosť celej zvukovej ukážky	2
Rozloženie nástrojov v priestore	0
Rozdiely vo frekvenčnom spektre	6
Rozdielny dozvuk	2

Počet nesprávnych odpovedí je v tomto prípade nižší, čo môže znamenať výraznejší rozdiel ako v prípade Otázky 1. Výsledky však stále naznačujú, že rozhodnúť medzi odpoveďami bolo náročné.

V prípade otázky 5 sa jedná o mixáž skladby *I'm Alright*, pričom mixáž prebehla so zaradením efektu Reverb a remixáž pomocou modelu `Test_model_reverb`. V tomto prípade je počuteľný výrazný rozdiel medzi cieľovým mixom a remixážou, čomu zodpovedajú aj výsledky uvedené v tabuľke 5.5.

Vo všetkých prípadoch bola správne určená odpoveď B.

Tab. 5.5: Odpovede na otázku 5

Otázka 5	Počet odpovedí
A	0
B	19
Náhodný výber	0
Neviem posúdiť	0
Hlasitosť jednotlivých nástrojov	9
Hlasitosť celej zvukovej ukážky	3
Rozloženie nástrojov v priestore	11
Rozdiely vo frekvenčnom spektre	12
Rozdielny dozvuk	4

V otázke 6 bola použitá opäť nahrávka skladby *Rachel*, pričom v tomto prípade bola mixovaná so zaradením efektu *Reverb* a remixovaná pomocou modelu `Test_model_reverb`. Výsledky sú zhrnuté v tabuľke 5.6 a naznačujú počuteľný rozdiel medzi mixom a remixom.

Tab. 5.6: Odpovede na otázku 6

Otázka 6	Počet odpovedí
A	2
B	17
Náhodný výber	0
Neviem posúdiť	0
Hlasitosť jednotlivých nástrojov	4
Hlasitosť celej zvukovej ukážky	0
Rozloženie nástrojov v priestore	6
Rozdiely vo frekvenčnom spektre	12
Rozdielny dozvuk	6

Otázka 7 bola postavená na mixáži bez zaradeného efektu *Reverb* a remixáži pomocou modelu `Test_model_filter`. Správnou odpoveďou bola možnosť B, ktorú označili štyria respondenti. Výsledky sú zhrnuté v tabuľke 5.7.

Vzhľadom na nízky počet správnych odpovedí a označené možnosti náhodný výber by sme mohli konštatovať, že v tomto prípade je predikcia siete najbližšie k cieľovému mixu.

V otázke 8 bola porovnávaná mixáž druhého úseku skladby *Ecstasy*, tentokrát so zaradeným efektom *Reverb* a remixáž prebiehala použitím modelu so zaradením efektu *Reverb*. Výsledky sú zhrnuté v tabuľke 5.8

Tab. 5.7: Odpovede na otázku 7

Otázka 7	Počet odpovedí
A	8
B	4
Náhodný výber	3
Neviem posúdiť	8
Hlasitosť jednotlivých nástrojov	3
Hlasitosť celej zvukovej ukážky	0
Rozloženie nástrojov v priestore	3
Rozdiely vo frekvenčnom spektre	7
Rozdielny dozvuk	1

Tab. 5.8: Odpovede na otázku 8

Otázka 8	Počet odpovedí
A	19
B	0
Náhodný výber	0
Neviem posúdiť	0
Hlasitosť jednotlivých nástrojov	4
Hlasitosť celej zvukovej ukážky	2
Rozloženie nástrojov v priestore	11
Rozdiely vo frekvenčnom spektre	8
Rozdielny dozvuk	13

V tomto prípade bola správna odpoveď opäť jednoznačne určená všetkými respondentmi.

V otázke 9 sa jednalo opäť o porovnanie mixov skladby *Rachel*, v tomto prípade druhého úseku skladby, so zaradením efektu *Reverb*. Výsledky sú zhrnuté v tabulke 5.9 a naznačujú počuteľný rozdiel medzi mixom a remixážou.

Správnou odpoveďou bolo B, označené vo značnej väčšine prípadov.

V poslednej otázke sa jednalo o mixáž skladby *I'm Alright* bey zaradeného efektu *Reverb*. Výsledky sú zhrnuté v tabuľke 5.10 a vyplýva z nich vyššia náročnosť určenia správnej odpovede, čo naznačuje vyššiu presnosť remixáže.

Tab. 5.9: Odpovede na otázku 9

Otázka 9	Počet odpovedí
A	1
B	17
Náhodný výber	0
Neviem posúdiť	1
Hlasitosť jednotlivých nástrojov	4
Hlasitosť celej zvukovej ukážky	2
Rozloženie nástrojov v priestore	9
Rozdiely vo frekvenčnom spektre	8
Rozdielny dozvuk	5

Tab. 5.10: Odpovede na otázku 10

Otázka 10	Počet odpovedí
A	11
B	5
Náhodný výber	2
Neviem posúdiť	3
Hlasitosť jednotlivých nástrojov	5
Hlasitosť celej zvukovej ukážky	1
Rozloženie nástrojov v priestore	2
Rozdiely vo frekvenčnom spektre	8
Rozdielny dozvuk	1

5.4 Zhrnutie výsledkov porovnania

Na základe vykonaných porovnaní je možné usúdiť, že oba modely predstavené v tejto bakalárskej práci fungujú v zmysle určovania takých parametrov signálových procesorov použitých pri mixáži zvuku, ktorých výsledok sa približuje k cieľovému mixu, ktorého parametre má za cieľ model odhadnúť. Z analýzy však vyplýva, že tieto parametre nemusia byť identické ako parametre použité pri mixáži cieľového mixu, avšak sú vedúce k mixu veľmi príbuznému cieľovému mixu. V prípade, že sú použité parametre určené natrénovaným modelom neurónovej siete predstaveným v tejto práci, s danou presnosťou sa remixáž neurónovej siete približuje cieľovému mixu.

Z objektívneho porovnania vyplýva, že model so zaradeným efektom *Reverb* dosahuje pri natrénovaní pomocou tréningového cyklu popísaného v 4.4 nižšiu presnosť odhadu ako model bez zaradeného efektu *Reverb*, čo pravdepodobne vyplýva z výraz-

ného zvýšenia komplexnosti a výpočetnej náročnosti problému spôsobenej pridaním určovanej váhy efektu, ktorá predstavuje impulznú odozvu o dĺžke 44100 vzorkov.

V rámci subjektívneho porovnania bol vykonaný porovnávací test vychádzajúci z princípu *ABX* testu. Vzhľadom na časové a materiálne obmedzenia sa nejednalo o reálny *ABX* test a vykonanie testu malo primárne demonšratívne účely. Vzhľadom na nedostatky testovacej metódy nie je možné výsledky testu považovať za výpovedné, avšak aj napriek tomu naznačujú potvrdenie výsledkov vyplývajúcich z objektívneho porovnávania.

Z celkových výsledkov dosiahnutých v objektívnom porovnávaní parametrov remixáže a cieľového mixu, v objektívnom porovnaní remixáže a cieľového mixu a čiastočne v subjektívom porovnaní je možné usúdiť, že modely predstavené v tejto práci je možné použiť na určenie takých parametrov mixáže, ktoré povedú k výsledku podobnému až identickému s cieľovým mixom. Modely však nie sú vhodné na presné určenie parametrov použitých pri mixáži cieľového mixu z dôvodov uvedených v kapitole 5.2.

Záver

Hlavným cieľom tejto bakalárskej práce bolo vytvoriť model neurónovej siete so špeciálnou štruktúrou pozostávajúcou z prvkov implementovaných pomocou knižnice *DDSP*, ktorá kombinuje prístup DSP (digitálneho spracovania signálu) s princípami strojového učenia. Podstatou realizácie modelu bolo zoradenie signálových procesorov implementovaných prostredníctvom *DDSP* do štruktúry podobnej štruktúre štandardne používanej pri mixáži zvuku, pričom jednotlivé signálové procesory vystupovali zároveň v zmysle vrstiev neurónovej siete, pričom parametre zoradených efektov reprezentovali váhy neurónovej siete. Cieľom takéhoto modelu je na základe znalosti vstupných stôp a cieľového mixu určiť jeho parametre.

Takto zostaveným modelom boli následne určované odhady parametrov použitých pri mixáži cieľových mixov slúžiacich v zmysle tréningových dát neurónovej siete. Cieľové mixy boli vytvorené mixážou pomocou signálových procesorov z knižnice *DDSP*. Jednalo sa o 5 skladieb, z ktorých boli vytvorené dve verzie, jedna bez zaradenia efektu *Reverb*, druhá so zaradením tohto efektu. Skladby spadali do žánrovo odlišných kategórií.

V rámci naplnenia cieľov práce boli vytvorené dva modely neurónovej siete, pričom jeden zodpovedal štruktúre bez implementácie efektu *Reverb*, v prípade druhého modelu bol efekt zaradený do jeho štruktúry. Oba predstavené modely sú schopné počas tréningového procesu na základe predpokladanej znalosti vstupných stôp a cieľového mixu určiť také parametre signálových procesorov, ktoré vedú k dosiahnutiu výsledného mixu blízkeho cieľovému mixu.

V rámci práce bola presnosť oboch modelov porovnávaná objektívnymi aj subjektívnymi metódami, pričom objektívne boli porovnávané odhadnuté parametre použité pri mixáži so skutočne použitými parametrami a výsledná remixáž s cieľovým mixom. Remixáž a cieľový mix boli zrovnávané aj prostredníctvom demonštratívneho subjektívneho testu založeného na porovnávaní dvoch zvukových vzoriek s referenčnou vzorkou. Vzhľadom na povahu vykonaného subjektívneho testovania nie je možné výsledky považovať za smerodajné. Aj napriek tomu výsledky získané subjektívnym porovnaním korešpondujú s výsledkami objektívneho porovnávania, z ktorého vyplýva, že model so zaradeným efektom *Reverb* dosahuje nižšej presnosti ako model bez zaradeného efektu.

Pri objektívnom zrovnávaní parametrov určených neurónovou sieťou s reálnymi parametrami cieľového mixu boli zistené značné odchýľky medzi parametrami určenými neurónovou sieťou a skutočnými parametrami, aj napriek tomu že výsledná remixáž pomocou určených parametrov nevykazovala výraznejší rozdiel oproti cieľovému mixu. Tento jav je pravdepodobne spôsobený tým, že je možné ddosiahnuť zvukovo identického výsledku pri rôznych nastaveniach parametrov použitých sig-

nálových procesorov.

V rámci bakalárskej práce sa podarilo naplniť stanovené ciele, a to implementáciu neurónovej siete pomocou signálových procesorov implementovaných pomocou knižnice *DDSP*, vytvorenie cieľových mixov ručnou mixážou pomocou efektov implementovaných v *DDSP* a následné vyhodnotenie presnosti modelov použitím objektívnych a subjektívnych metód.

Zlepšenie výsledkov práce by mohlo byť dosiahnuté lepšie navrhnutým tréningovým procesom neurónovej siete, dostupnosťou vyššieho výpočetného výkonu a vykonaním vyššieho počtu iterácií v priebehu tréningu neurónovej siete a lepšie realizovanou subjektívnou porovnávacou metódou.

Literatúra

[1] COLONEL, Joseph T.; REISS, Joshua. Reverse engineering of a recording mix with differentiable digital signal processinga). *The Journal of the Acoustical Society of America*. 2021, roč. 150, č. 1, s. 608–619. ISSN 0001-4966. Dostupné z DOI: 10.1121/10.0005622.

[2] PATTERSON, Josh; GIBSON, Adam. *Deep Learning*. O'Reilly Media, 2017. ISBN 978-1-491-91425-0.

[3] VASILEV, Ivan; SLATER, Daniel; SPACAGNA, Gianmario; ROELANTS, Peter; ZOCCA, Valentino. *Python Deep Learning*. Second. Packt Publishing, 2019. ISBN 978-1-78934-846-0.

[4] TURING. *What Is the Necessity of Bias in Neural Networks?* 2023. Dostupné tiež z: https://www.turing.com/kb/necessity-of-bias-in-neural-networks.

[5] BENOVIČ, Erik. *Neuronové síte a jejich aplikace*. Brno, 2021. Dostupné tiež z: https://is.muni.cz/th/s93mf/thesis_final.pdf?kod=BKF_OSFI.

[6] DVOŘÁKOVÁ, Gabriela. *Použití DNN pro analýzu trojúhelníkových sítí v geometrické morfometrii*. Praha, 2018. Dostupné tiež z: https://dspace.cuni.cz/bitstream/handle/20.500.11956/100954/130230907.pdf?sequence=1&isAllowed=y. Bakalárska práca. Univerzita Karlova.

[7] CAMPESATO, Oswald. *Artificial Intelligence, Machine Learning and Deep Learning*. Mercury Learning a Information, 2020. ISBN 978-1-68392-467-8.

[8] DELUA, Julliana. *Supervised versus unsupervised learning: What's the difference?* [online]. 2024. [cit. 2024-05-26]. Dostupné z : https://www.ibm.com/think/topics/supervised-vs-unsupervised-learning.

[9] BHAT, Rauf. *Gradient Descent With Momentum*. 2020. Dostupné tiež z: https://towardsdatascience.com/gradient-descent-with-momentum-59420f626c8f. [cit. 2024-05-13].

[10] BUDUMA, Nikhil; LASCACIO, Nicholas. *Fundamentals of Deep Learning*. 2nd. O'Reilly Media, 2017. ISBN 978-1-491-92561-4.

[11] *Root-Mean-Square* [online]. [cit. 2024-05-26]. Dostupné z : https://mathworld.wolfram.com/.

[12] FARKAŠ, Martin. *Realizace neuronové sítě s využitím grafických procesorů*. Plzeň, 2019. Bakalárska práca. Západočeská univerzita v Plzni.

[13] KINGMA, Diederik P.; BA, Jimmy. *Adam: A Method for Stochastic Optimization*. 2017. Dostupné z arXiv: 1412.6980 [cs.LG].

[14] ENGEL, Jesse; HANTRAKUL, Lamtharn (Hanoi); GU, Chenjie; ROBERTS, Adam. DDSP: Differentiable Digital Signal Processing. In: *International Conference on Learning Representations*. 2020. Dostupné tiež z: https://openreview.net/forum?id=B1x1ma4tDr.

[15] SYROVÝ, Václav. *Hudební zvuk*. Druhá. Akademie múzických umění, 2014. ISBN 978-80-7331-323-4.

[16] *DDSP Processor Demo*. 2020. Dostupné tiež z: https://colab.research.google.com/github/magenta/ddsp/blob/main/ddsp/colab/tutorials/0_processor.ipynb#scrollTo=LSvRsjJUVxOA. [cit. 2023-12-05].

[17] *DDSP Synths and Effects*. 2020. Dostupné tiež z: https://colab.research.google.com/github/magenta/ddsp/blob/main/ddsp/colab/tutorials/1_synths_and_effects.ipynb#scrollTo=mkjWUGpZ95Mr. [cit. 2023-12-05].

[18] SENIOR, Mike. *Mixing Secrets For The Small Studio - Additional Resources* [Dostupné z]. [cit. 2023-12-13]. Dostupné z : https://cambridge-mt.com/ms/mtk/. Online.

[19] MAGENTA PROJECT. *DDSP: Differentiable Digital Signal Processing* [https://github.com/magenta/ddsp]. 2020. Online; accessed 2023-12-05.

[20] *tf.random_normal_initializer* [Online]. n.d. [cit. 2024-05-16]. Dostupné z : https://www.tensorflow.org/api_docs/python/tf/random_normal_initializer. TensorFlow.

[21] *Tf.random.uniform* [Online]. TensorFlow, 2024. Dostupné tiež z: https://www.tensorflow.org/api_docs/python/tf/random/uniform.

[22] VÄLIMÄKI, Vesa; REISS, Joshua D. All About Audio Equalization: Solutions and Frontiers. *Applied Sciences*. 2016, roč. 6, č. 5, s. 129. Dostupné z DOI: 10.3390/app6050129.

[23] VLČKOVÁ, Irena. *Analýza stochastických procesů v elektronických součástkách*. Praha, 2009.

[24] MCLAUGHLIN, Sean. *Mixing with iZotope* [https://downloads.izotope.com/guides/iZotope-Mixing-Guide-Principles-Tips-Techniques.pdf]. iZotope, 2014. Online; accessed 2023-12-05.

[25] IZHAKI, Roey. *Mixing Audio: Concepts, Practices and Tools*. Focal Press, 2008. ISBN 978-0-240-52068-1.

[26] DEVELOPERS, NumPy. *numpy.reshape()* [online]. 2022. [cit. 2023-12-13]. Dostupné z : https://numpy.org/doc/stable/reference/generated/numpy.reshape.html#numpy.reshape. Online.

[27] *Forms* [online]. 2024. [cit. 2024-05-23]. Dostupné z : https://colab.research.google.com/notebooks/forms.ipynb.

[28] DEVELOPERS, NumPy. *numpy.vstack()* [online]. 2022. [cit. 2023-12-13]. Dostupné z : https://numpy.org/doc/stable/reference/generated/numpy.vstack.html. Online.

[29] DISK MAGAZÍN. *Rozdíly mezi mikrofonními technikami XY, ORTF a NOS Pro jaký stereo obraz se hodí více či méně*. 2024. Dostupné tiež z: https://magazin.disk.cz/cs/rozdily-mez-mikrofonnimi-technikami-xy-ortf-a-nos. Accessed: 2024-05-23.

[30] *API*. 2024. Dostupné tiež z: https://www.britannica.com/technology/API. Accessed: 2024-05-23.

[31] *The Model Class* [online]. 2023. [cit. 2024-05-23]. Dostupné z : https://keras.io/api/models/model/. Online.

[32] *Layer Weight Initializers* [online]. 2024. [cit. 2024-05-23]. Dostupné z : https://keras.io/api/layers/initializers/. Online.

[33] *Absolute Error* [online]. 2024. [cit. 2024-05-27]. Dostupné z : https://mathworld.wolfram.com/.

[34] INTERNATIONAL TELECOMMUNICATION UNION. *Algorithms to measure audio programme loudness and true-peak audio level*. 2023. Tech. spr., ITU-R BS.1770-5. ITU.

[35] STEINMETZ, Christian J.; REISS, Joshua D. pyloudnorm: A simple yet flexible loudness meter in Python. In: *150th AES Convention*. 2021.

[36] *Focus: The Audio Vectorscope* [online]. RTW GmbH. [cit. 2024-05-27]. Dostupné z : https://www.rtw.com/en/blog/focus-the-audio-vectorscope.html.

[37] MCFEE, Brian; RAFFEL, Colin; LIANG, Dawen; ELLIS, Daniel; MCVICAR, Matt; BATTENBERG, Eric; NIETO, Oriol. librosa: Audio and Music Signal Analysis in Python. In: 2015, s. 18–24. Dostupné z DOI: 10.25080/Majora-7b98e3ed-003.

[38] INTERNATIONAL TELECOMMUNICATION UNION. *Methods for the Subjective Assessment of Small Impairments in Audio Systems*. 2nd. [B.r.]. ITU-R Recommendation BS.1116.

Zoznam symbolov a skratiek

DSP	číslicové spracovanie signálu – Digital Signal Processing
DDSP	Diferencovateľné digitálne spracovanie signálu – Differentiable digital signal processing–knižnica v jazyku Python zameraná na strojové učenie pomocou interpretovateľných signálových procesorov
LTI	Lineárny, časovo invariantný systém – Linear time-invariant
FIR	konečná impulzná odozva – Finite Impulse Response

Zoznam príloh

A Ilustračné výstrižky v jazyku *Python*

A.1 Funkcia modelujúca frekvenčnú odozvu

Výpis A.1: Implementácia metódy `design_bandpass_response`

```
def design_bandpass_response(band_gain, num_points=1025,
f_c=[31, 63, 125, 250, 500, 1000, 2000, 4000, 8000,
16000], sampling_rate=44100):

  nyquist_freq = sampling_rate / 2
  freqs = np.linspace(0, 1, num_points) * nyquist_freq
  magnitude = np.zeros(freqs.shape)

  for j in range(0, 10):
  # urcenie dolneho a horneho medzneho kmitoctu
    f_min = f_c[j] / np.sqrt(2)
    f_max = f_c[j] * np.sqrt(2)
  #vypocet zodpovedajuceho vzorku frekvencnej odozvy
    low_cut = int((f_min) * (1025 / nyquist_freq))
    high_cut = int((f_max) * (1025 / nyquist_freq))
#obmedzenie maximalnej hodnoty kmitoctu frekvencnej
#odozvy
    if high_cut >= 1025:
    high_cut = 1024

# vytvorenie kmitoctovej odozvy
  for i in range(low_cut, high_cut):
    magnitude[i] = band_gain[j]
#uprava tvaru na tvar vyzadovany na vstupe efektu
#ddsp.effect.FIRfilter()
  magnitude = np.reshape(magnitude, (1, 1025))
  return magnitude
```

A.2 Zdrojový kód modelov neurónovej siete v jazyku Python

A.2.1 Model `Test_model_filter`

Výpis A.2: Implementácia triedy `Test_model_filter`

```
1
2  class Test_model_filter(tf.keras.Model):
3    def __init__(self, num_of_chan):
4      super().__init__()
5
6      #Vytvorenie struktury siete v zavislosti na pocte
7      #kanalov
8      self._num_of_chan = num_of_chan #definicia poctu
9      kanalov
10     self._gains = []
11     self._eqs = []
12     self._pans = []
13
14     #Inicializacia signalovych procesorov v zavislosti
15     #na pocte kanalov
16     for i in range(0, self._num_of_chan):
17       self._gains.append(model_layers.Gain
18       (trainable=True))
19       self._eqs.append(model_layers.EQ
20       (trainable=True))
21       self._pans.append(model_layers.Pan
22       (trainable=True))
23       self._add = ddsp.processors.Add("add")
24
25   #definicia dopredneho kroku
26   def call(self, inputs):
27
28     added_L = self._pans[0].get_signal(self._gains[0].
29     get_signal(self._eqs[0].get_signal(inputs[0])))[0]
30
31     added_R = self._pans[0].get_signal(self._gains[0].
32     get_signal(self._eqs[0].get_signal(inputs[0])))[1]
```

```
33
34      #spracovanie a scitanie jednotlivych kanalov
35      for i in range(1, self._num_of_chan):
36        added_L = self._add.get_signal(added_L,
37        self._pans[i].get_signal(
38        self._gains[i].get_signal(
39        self._eqs[i].get_signal(inputs[i])))[0])
40
41        added_R = self._add.get_signal(added_R,
42        self._pans[i].get_signal(
43        self._gains[i].get_signal(
44        self._eqs[i].get_signal(inputs[i])))[1])
45
46      return [added_L, added_R]
```

A.2.2 Model `Test_model_reverb`

Výpis A.3: Implementácia triedy `Test_model_reverb`

```
1
2  class Test_model_reverb(tf.keras.Model):
3    def __init__(self, num_of_chan):
4      super().__init__()
5
6      #vytvorenie struktury siete v zavislosti na pocte
7      #definovanych vstupnych kanalov
8      self._num_of_chan = num_of_chan
9      self._gains = []
10     self._eqs = []
11     self._pans = []
12     self._sends = []
13     for i in range(0, self._num_of_chan):
14       self._gains.append(model_layers.Gain(
15       trainable=True))
16       self._eqs.append(model_layers.EQ(
17       trainable=True))
18       self._pans.append(model_layers.Pan(
19       trainable=True))
20       self._sends.append(model_layers.Gain(
21       trainable=True))
22       self._add = ddsp.processors.Add("add")
23
24     #definicia zbernice reverbu
25     self._reverb_L = ddsp.effects.Reverb(
26     reverb_length=44100, add_dry=False, trainable=True)
27     self._reverb_L.build(input)
28     self._reverb_R = ddsp.effects.Reverb(
29     reverb_length=44100, add_dry=False, trainable=True)
30     self._reverb_R.build(input)
31
32   def call(self, inputs):
33
34     #definicia dopredneho kroku
35     panned_L = self._pans[0].get_signal(
36      self._gains[0].get_signal(
```

```
37        self._eqs[0].get_signal(inputs[0])))[0]
38
39      panned_R = self._pans[0].get_signal(
40        self._gains[0].get_signal(
41        self._eqs[0].get_signal(inputs[0])))[1]
42
43      verbed_L = self._sends[0].get_signal(
44        self._pans[0].get_signal(
45        self._gains[0].get_signal(
46        self._eqs[0].get_signal(inputs[0])))[0])
47
48      verbed_R = self._sends[0].get_signal(
49        self._pans[0].get_signal(
50        self._gains[0].get_signal(
51        self._eqs[0].get_signal(inputs[0])))[1])
52
53      added_L = panned_L
54      added_R = panned_R
55
56      for i in range(1, self._num_of_chan):
57        panned_L = self._pans[i].get_signal(
58          self._gains[i].get_signal(
59          self._eqs[i].get_signal(inputs[i])))[0]
60
61        panned_R = self._pans[i].get_signal(
62          self._gains[i].get_signal(
63          self._eqs[i].get_signal(inputs[i])))[1]
64
65        #vypocet stereofonneho mixu
66        added_L = self._add.get_signal(added_L, panned_L)
67        added_R = self._add.get_signal(added_R, panned_R)
68
69        #vypocet mixu na zbernici reverbu
70        verbed_R = self._add.get_signal(verbed_R,
71          self._sends[i].get_signal(panned_R))
72
73        verbed_L = self._add.get_signal(verbed_L,
74          self._sends[i].get_signal(panned_L))
75
```

```
76      #spracovanie signalu efektom reverb
77      verbed_R = self._reverb_R.get_signal(verbed_R,
78       self._reverb_R._ir)
79      verbed_L = self._reverb_L.get_signal(verbed_L,
80       self._reverb_L._ir)
81
82      #zmiesanie signalu odoberaneho za efektom Pan so
83      #signalom spracovanym efektom Reverb
84      added_L = self._add.get_signal(added_L, verbed_L)
85      added_R = self._add.get_signal(added_R, verbed_R)
86
87      return [added_L, added_R]
```

B Parametre efektov použitých pri mixáži

B.1 Mix piesne Ecstasy

Tab. B.1: Tabuľka parametrov použitých pri mixáži piesne Ecstasy (parametre f1 až f10 určujú zosilnenia jednotlivých frekvenčných pásmiem oktávového ekvalizéru)

	Drums	Bass	Synth L	Synth R	Vox
f1	0,82	0,92	0,02	0,02	0,02
f2	0,95	0,92	0,02	0,02	0,38
f3	0,96	0,92	0,48	0,48	0,94
f4	0,96	0,71	0,67	0,67	0,93
f5	0,87	0,46	0,86	0,86	0,94
f6	0,84	0,28	0,96	0,96	0,95
f7	0,88	0,15	1	1	1
f8	0,99	0,09	1	1	1
f9	0,99	0,06	1	1	0,95
f10	0,99	0,03	1	1	0,91
gain	0,83	0,71	0,74	0,73	0,89
pan	0	0	0,51	-0,51	0
send	0,24	0	0,32	0,29	0,28

B.2 Mix piesne I'm Alright

Tab. B.2: Tabuľka parametrov použitých pri mixáži piesne I'm Alright

	Drums	Bass	Piano	Elec. Guitar	Lead Vox	Backing Vox	Acous. Guitar
f1	0,82	0,92	0,02	0,02	0,02	0,02	0,02
f2	0,95	0,92	0,02	0,02	0,38	0,38	0,53
f3	0,96	0,92	0,48	0,32	0,94	0,65	0,94
f4	0,96	0,71	0,67	0,32	0,93	0,78	0,93
f5	0,87	0,46	0,86	0,69	0,94	0,94	0,94
f6	0,84	0,28	0,96	0,88	0,95	0,95	0,95
f7	0,88	0,15	1	0,96	1	1	1
f8	0,99	0,09	1	1	1	1	1
f9	0,99	0,06	1	0,79	0,95	0,95	0,95
f10	0,99	0,03	1	0,92	0,91	0,91	0,91
gain	0,94	0,71	0,74	0,84	1	0,8	0,94
pan	0	0	0	-0,26	0	0	0,41
send	0	0	0	0,41	0,33	0,32	0,23

B.3 Mix piesne Ubiquitous

Tab. B.3: Tabuľka parametrov použitých pri mixáži piesne Ubiquitous

	Drums	Bass	Synth 1	Synth 2	Lead Synth	SFX
f1	0,82	0,92	0,02	0,02	0,02	0,02
f2	0,95	0,92	0,02	0,38	0,02	0,38
f3	0,96	0,92	0,32	0,94	0,48	0,65
f4	0,96	0,71	0,69	0,93	0,67	0,78
f5	0,87	0,46	0,88	0,94	0,86	0,94
f6	0,84	0,28	0,96	0,95	0,96	0,95
f7	0,88	0,15	1	1	1	1
f8	0,99	0,09	0,79	1	1	1
f9	0,99	0,06	0,92	0,95	1	0,95
f10	0,99	0,03	1	0,91	1	0,91
gain	1	0,78	0,98	1	0,74	0,8
pan	0	0	-0,26	0,56	0	0
send	0	0	0,32	0,29	0,22	0,47

B.4 Mix skladby Jesu bleibet meine Freude

Tab. B.4: Tabuľka parametrov použitých pri mixáži piesne Jesu bleibet meine Freude

	Strings R	Strings L	ORTF R	ORTF L	Spot mic
f1	0,14	0,82	0,02	0,02	0,02
f2	0,93	0,95	0,75	0,02	0,38
f3	0,92	0,96	0,82	0,32	0,94
f4	0,92	0,87	0,91	0,69	0,93
f5	0,92	0,84	0,92	0,88	0,94
f6	0,95	0,88	0,96	0,96	0,95
f7	0,79	0,99	1	1	1
f8	0,62	0,99	1	0,97	1
f9	0,48	0,99	1	0,97	0,95
f10	0,34	0,99	1	1	0,91
gain	0,8	0,93	0,95	0,98	1
pan	0,13	-0,2	-0,13	0,21	-0,13
send	0,13	0,17	0,22	0,19	0,13

B.5 Mix skladby Rachel

Tab. B.5: Tabuľka parametrov použitých pri mixáži piesne Rachel

	Cello	Bass	Guitar	Vocal Lead	Percs	Ukulele	Viola
f1	0,39	0,94	0,02	0,02	0,02	0,02	0,02
f2	0,68	0,95	0,02	0,87	0,38	0,38	0,53
f3	0,99	0,96	0,48	0,84	0,59	0,65	0,94
f4	0,99	0,96	0,67	0,85	0,76	0,93	0,93
f5	0,98	0,76	0,86	0,93	0,87	0,98	0,94
f6	0,8	0,67	0,96	0,96	0,95	0,98	0,95
f7	0,61	0,54	1	1	1	1	1
f8	0,48	0,3	1	0,79	1	1	1
f9	0,29	0,16	1	0,92	0,95	0,95	0,95
f10	0,15	0,05	1	1	0,91	0,91	0,91
gain	0,77	0,8	0,9	0,98	0,68	0,98	0,93
pan	0	0	0,27	0	0	-0,3	-0,2
send	0,14	0	0	0,12	0,22	0,32	0,23

C Porovnanie parametrov určených neurónovou sieťou s parametrami cieľového mixu

Tab. C.1: Porovnanie parametrov skladby Ubiquitous určených modelom bez zaradenia efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,9318	1	-0,0682
Gain 2	0,8627	0,78	0,0827
Gain 3	0,7198	0,74	-0,0202
Gain 4	0,7866	0,98	-0,1934
Gain 5	0,7308	1	-0,2692
Gain 6	0,4003	0,8	-0,3997
Pan 1	0,0028	0,01	-0,0072
Pan 2	0,0006	0	0,0006
Pan 3	0,0404	0	0,0404
Pan 4	-0,2646	-0,26	-0,0046
Pan 5	0,5569	0,29	0,2669
Pan 6	0,2634	0	0,2634

Tab. C.2: SNR parametrov efektov EQ bez zaradeného efektu Reverb

	SNR [dB]
EQ 1	2,45
EQ 2	-11,71
EQ 3	5,81
EQ 4	5,8
EQ 5	6,18
EQ 6	4,88

Tab. C.3: Porovnanie parametrov skladby Ubiquitous určených modelom so zaradením efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,4354	1	-0,5646
Gain 2	0,9519	0,78	0,1719
Gain 3	0,3521	0,74	-0,3879
Gain 4	0,6922	0,98	-0,2878
Gain 5	0,4748	1	-0,5252
Gain 6	0,4181	0,8	-0,3819
Pan 1	0,4648	0,01	0,4548
Pan 2	-0,0013	0	-0,0013
Pan 3	0,453	0	0,453
Pan 4	0,1446	-0,26	0,4046
Pan 5	1,0136	0,29	0,7236
Pan 6	0,5658	0	0,5658
Send 1	0,7312	0,01	0,7212
Send 2	0,0012	0	0,0012
Send 3	0,5311	0,22	0,3111
Send 4	0,1428	0,32	-0,1772
Send 5	0,4791	0,29	0,1891
Send 6	0,0192	0,47	-0,4508

Tab. C.4: SNR parametrov efektov EQ a Reverb so zaradením efektu Reverb

	SNR [dB]
EQ 1	4,57
EQ 2	-11,36
EQ 3	5,39
EQ 4	5,23
EQ 5	5,52
EQ 6	4,92
Reverb L	-0,91
Reverb R	-5,94

Tab. C.5: Porovnanie parametrov skladby Ecstacy určených modelom bez zaradenia efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,7003	0,83	-0,1297
Gain 2	0,6726	0,71	-0,0374
Gain 3	0,8239	0,74	0,0839
Gain 4	1,081	0,73	0,351
Gain 5	1,3223	0,89	0,4323
Pan 1	0	0	0
Pan 2	0	0	0
Pan 3	0,51	0,51	0
Pan 4	-0,51	-0,51	0
Pan 5	0	0	0

Tab. C.6: SNR parametrov efektov EQ bez zaradeného efektu Reverb

	SNR [dB]
EQ 1	5,64
EQ 2	-10,84
EQ 3	5,88
EQ 4	5,3
EQ 5	4,76

Tab. C.7: Porovnanie parametrov skladby Ecstacy určených modelom so zaradením efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,4668	0,83	-0,3632
Gain 2	0,3783	0,71	-0,3317
Gain 3	1,044	0,74	0,304
Gain 4	0,4333	0,73	-0,2967
Gain 5	1,2669	0,89	0,3769
Pan 1	0,3105	0	0,3105
Pan 2	0,2963	0	0,2963
Pan 3	0,7024	0,51	0,1924
Pan 4	0,2498	-0,51	0,7598
Pan 5	0,2528	0	0,2528
Send 1	0,9329	0,24	0,6929
Send 2	0,8263	0	0,8263
Send 3	0,4292	0,32	0,1092
Send 4	0,9747	0,29	0,6847
Send 5	0,3891	0,28	0,1091

Tab. C.8: SNR parametrov efektov EQ a Reverb so zaradením efektu Reverb

	SNR [dB]
EQ 1	5,69
EQ 2	-11,68
EQ 3	4,81
EQ 4	4,73
EQ 5	4,99
Reverb L	-0,72
Reverb R	-2,72

Tab. C.9: Porovnanie parametrov skladby Jesu bleibet meine Freude určených modelom bez zaradenia efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,8513	0,8	0,0513
Gain 2	1,1264	0,93	0,1964
Gain 3	0,997	0,95	0,047
Gain 4	1,023	0,98	0,043
Gain 5	1,4099	1	0,4099
Pan 1	0,2798	0,28	-0,0002
Pan 2	-0,1989	-0,2	0,0011
Pan 3	-0,1298	-0,13	0,0002
Pan 4	0,2101	0,21	0,0001
Pan 5	-0,1298	-0,13	0,0002

Tab. C.10: SNR parametrov efektov EQ bez zaradeného efektu Reverb

SNR [dB]	
EQ 1	4,31
EQ 2	6,05
EQ 3	5,68
EQ 4	5,27
EQ 5	5,24

Tab. C.11: Porovnanie parametrov skladby Jesu bleibet meine Freude určených modelom so zaradením efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,4604	0,8	-0,3396
Gain 2	0,8195	0,93	-0,1105
Gain 3	0,6569	0,95	-0,2931
Gain 4	0,3019	0,98	-0,6781
Gain 5	0,8885	1	-0,1115
Pan 1	0,5812	0,28	0,3012
Pan 2	0,2071	-0,2	0,4071
Pan 3	-0,0888	-0,13	0,0412
Pan 4	0,5333	0,21	0,3233
Pan 5	-0,05321	-0,13	0,07679
Send 1	0,6819	0,13	0,5519
Send 2	0,4643	0,17	0,2943
Send 3	0,3745	0,22	0,1545
Send 4	0,9802	0,19	0,7902
Send 5	0,3797	0,13	0,2497

Tab. C.12: SNR parametrov efektov EQ a Reverb so zaradením efektu Reverb

SNR [dB]	
EQ 1	3,55
EQ 2	4,85
EQ 3	5,08
EQ 4	4,99
EQ 5	4,81
Reverb L	-1,83
Reverb R	-4,91

Tab. C.13: Porovnanie parametrov skladby Rachel určených modelom bez zaradenia efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	1,126	0,77	0,356
Gain 2	0,9209	0,8	0,1209
Gain 3	1,1772	0,9	0,2772
Gain 4	1,2565	0,98	0,2765
Gain 5	0,9992	0,68	0,3192
Gain 6	1,546	0,98	0,566
Gain 7	1,638	0,93	0,708
Pan 1	0	0	0
Pan 2	0	0	0
Pan 3	0,2697	0,27	-0,0003
Pan 4	0	0	0
Pan 5	0,01	0	0,01
Pan 6	-0,3007	-0,3	-0,0007
Pan 7	-0,2	-0,2	0

Tab. C.14: SNR parametrov efektov EQ bez zaradeného efektu Reverb

SNR [dB]	
EQ 1	-0,75
EQ 2	-4,58
EQ 3	4,99
EQ 4	4,62
EQ 5	4,98
EQ 6	5,02
EQ 7	5,23

Tab. C.15: Porovnanie parametrov skladby Rachel určených modelom so zaradením efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,6924	0,77	-0,0776
Gain 2	0,8563	0,8	0,0563
Gain 3	0,533	0,9	-0,367
Gain 4	0,7376	0,98	-0,2424
Gain 5	0,4212	0,68	-0,2588
Gain 6	1,4184	0,98	0,4384
Gain 7	0,8778	0,93	-0,0522
Pan 1	0,2702	0	0,2702
Pan 2	0,3032	0	0,3032
Pan 3	0,5558	0,27	0,2858
Pan 4	0,4752	0	0,4752
Pan 5	0,4046	0	0,4046
Pan 6	-0,3088	-0,3	-0,0088
Pan 7	-0,0699	-0,2	0,1301
Send 1	0,2927	0,14	0,1527
Send 2	0,4759	0	0,4759
Send 3	0,3145	0	0,3145
Send 4	0,7996	0,12	0,6796
Send 5	0,5982	0,22	0,3782
Send 6	0,0213	0,32	-0,2987
Send 7	0,0793	0,23	-0,1507

Tab. C.16: SNR parametrov efektov EQ a Reverb so zaradením efektu Reverb

SNR [dB]	
EQ 1	-0,9
EQ 2	-1,14
EQ 3	-0,43
EQ 4	-1,34
EQ 5	-0,81
EQ 6	-0,88
EQ 7	-0,67
Reverb L	-0,64
Reverb R	-4,38

Tab. C.17: Porovnanie parametrov skladby I'm Alright určených modelom bez zaradenia efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,9557	0,94	0,0157
Gain 2	0,7016	0,71	-0,0084
Gain 3	0,7888	0,74	0,0488
Gain 4	0,735	0,84	-0,105
Gain 5	1,1696	1	0,1696
Gain 6	0,2966	0,8	-0,5034
Gain 7	-0,004	0,94	-0,944
Pan 1	-0,0023	0	-0,0023
Pan 2	-0,0004	0	-0,0004
Pan 3	-0,2599	0	-0,2599
Pan 4	0,0384	-0,26	0,2984
Pan 5	0	0	0
Pan 6	0,0023	0	0,0023
Pan 7	0,4642	0,41	0,0542

Tab. C.18: SNR parametrov efektov EQ bez zaradeného efektu Reverb

	SNR [dB]
EQ 1	-14,08
EQ 2	4,69
EQ 3	5,47
EQ 4	5,55
EQ 5	5,13
EQ 6	1,31
EQ 7	5,05

Tab. C.19: Porovnanie parametrov skladby I'm Alright určených modelom so zaradením efektu Reverb

	Odhad	Cieľ	Δ
Gain 1	0,6924	0,94	-0,94
Gain 2	0,8563	0,71	-0,71
Gain 3	0,5331	0,74	-0,74
Gain 4	0,7378	0,84	-0,84
Gain 5	0,4212	1	-1
Gain 6	1,4184	0,8	-0,8
Gain 7	0,8778	0,94	-0,94
Pan 1	0,2702	0	0
Pan 2	0,3032	0	0
Pan 3	0,5558	0	0
Pan 4	-0,4752	-0,26	0,26
Pan 5	0,4046	0	0
Pan 6	-0,3088	0	0
Pan 7	-0,0698	0,41	-0,41
Send 1	0,2926	0	0
Send 2	0,4759	0	0
Send 3	0,3145	0	0
Send 4	0,7995	0,41	-0,41
Send 5	0,5981	0,33	-0,33
Send 6	0,0213	0,32	-0,32
Send 7	0,0794	0,23	-0,23

Tab. C.20: SNR parametrov efektov EQ a Reverb so zaradením efektu Reverb

SNR [dB]	
EQ 1	-13,07
EQ 2	4,7
EQ 3	5,38
EQ 4	5,4
EQ 5	6,61
EQ 6	1,63
EQ 7	2,78
Reverb L	-1,45
Reverb R	-3,62

D Objektívne porovnanie remixáží s cieľovými mixami

Vysvetlivky k symbolom použitým v tabuľkách: LUFS r označuje hlasitosť remixáže, LUFS t hlasitosť cieľového mixu, ΔLUFS označuje rozdiel hlasitostí remixáže a cieľového mixu. LUFS L r označuje hlasitosť ľavého kanála remixáže, LUFS L t hlasitosť ľavého kanála cieľového mixu a ΔLUFS L rozdiel týchto dvoch hodnôt. LUFS R r označuje hlasitosť pravého kanála remixáže, LUFS R t hlasitosť pravého kanála cieľového mixu a ΔLUFS R ich rozdiel. SNR [dB] označuje odstup signál-šum vypočítaný na celom zvukovom súbore, SNR L [dB] označuje odstup signál-šum vypočítaný zo spektrogramov ľavého kanála a SNR R [dB] odstup signál-šum vypočítaný zo sprektrogramov pravého kanála.

D.1 Porovnanie remixáže s cieľovým mixom na úseku, na ktorom boli trénované algoritmy

Tab. D.1: Mix skladby Ecstasy

	Ecstacy (filter)	Ecstacy (reverb)
LUFS r	-24,946	-24,677
LUFS t	-24,945	-24,404
ΔLUFS	-0,001	-0,273
LUFS L r	-27,396	-26,919
LUFS L t	-27,395	-26,813
ΔLUFS L	-0,001	-0,105
LUFS R r	-28,600	-28,621
LUFS R t	-28,599	-28,111
ΔLUFS R	-0,001	-0,510
SNR [dB]	36,84	15,54
SNR L [dB]	24,09	20,01
SNR R [dB]	24,35	22,22

Tab. D.2: Mix skladby I'm Alright

	I'm Alright (filter)	I'm Alright (reverb)
LUFS r	-23,216	-22,314
LUFS t	-23,213	-22,046
ΔLUFS	-0,003	-0,269
LUFS L r	-26,637	-25,914
LUFS L t	-26,634	-25,683
ΔLUFS L	-0,003	-0,232
LUFS R r	-25,852	-24,805
LUFS R t	-25,849	-24,508
ΔLUFS R	-0,003	-0,297
SNR [dB]	34,31	14,11
SNR L [dB]	20,70	18,34
SNR R [dB]	20,78	20,15

Tab. D.3: Mix skladby Ubiquitous

	Ubiquitous (filter)	Ubiquitous (reverb)
LUFS r	-15,685	-16,036
LUFS t	-15,682	-15,769
ΔLUFS	-0,003	-0,267
LUFS L r	-18,328	-18,681
LUFS L t	-18,331	-18,555
ΔLUFS L	0,003	-0,127
LUFS R r	-19,096	-19,445
LUFS R t	-19,086	-19,015
ΔLUFS R	-0,011	-0,429
SNR [dB]	35,83	16,54
SNR L [dB]	33,70	21,35
SNR R [dB]	33,74	22,30

Tab. D.4: Mix skladby Jesu bleibet meine Freude

	Jesu (filter)	Jesu (reverb)
LUFS r	-28,204	-28,199
LUFS t	-28,198	-28,117
ΔLUFS	-0,006	-0,081
LUFS L r	-30,918	-30,758
LUFS L t	-30,912	-30,719
ΔLUFS L	-0,006	-0,039
LUFS R r	-31,484	-31,754
LUFS R t	-31,477	-31,620
ΔLUFS R	-0,007	-0,134
SNR [dB]	34,88	21,45
SNR L [dB]	28,48	25,76
SNR R [dB]	27,98	23,41

Tab. D.5: Mix skladby Rachel

	Rachel (filter)	Rachel (reverb)
LUFS r	-28,543	-28,197
LUFS t	-28,540	-28,104
ΔLUFS	-0,003	-0,094
LUFS L r	-31,611	-30,986
LUFS L t	-31,608	-30,890
ΔLUFS L	-0,003	-0,096
LUFS R r	-31,496	-31,487
LUFS R t	-31,493	-31,308
ΔLUFS R	-0,004	-0,179
SNR [dB]	34,35	19,51
SNR L [dB]	20,52	20,31
SNR R [dB]	19,58	19,33

Tab. D.6: Porovnanie remixáží skladieb na úseku, na ktorom nebola neurónová sieť trénovaná

	ΔLUFS	ΔLUFS L	ΔLUFS R	SNR [dB]	SNR L [dB]	SNR R [dB]
Ecstacy (filter)	0,0004	0,0001	0,0007	36,43	25,07	25,33
Ecstacy (reverb)	-0,3708	-0,1634	-0,5934	13,40	21,12	22,65
Rachel (filter)	0,0021	0,0027	0,0015	33,754	22,16	21,38
Rachel (reverb)	-0,162	-0,119	-0,150	15,86	22,09	21,67

E Odkaz na zvukové súbory vo formáte .wav

https://drive.google.com/drive/folders/1Hi_ztdFh7wKvOl94UOmi9NTNvICRB5U1?usp=sharing

F Obsah elektronickej prílohy

- `/` ..... Koreňový adresár priloženého archívu
  - `Programy v jazyku Python.zip` ..... komprimovaný adresár s programami použitými v rámci práce
  - `Spektrogramy.zip` ..... komprimovaný adresár so spektrogramami remixáží a cieľových mixov
  - `Vektorskopy.zip` ... komprimovaný adresár s vektorskopmi remixáží a cieľových mixov