VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO PODPORU CVIČENÍ KLAVÍRNÍCH SKLADEB

TOOL FOR PIANO PRACTICING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. NIKITA USTINOV**
AUTHOR

VEDOUCÍ PRÁCE — **Dr. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání diplomové práce

Student:	**Ustinov Nikita, Bc.**
Program:	Informační technologie
Obor:	Inteligentní systémy
Název:	**Nástroj pro podporu cvičení klavírních skladeb** **Tool for Piano Practising**
Kategorie:	Umělá inteligence

Zadání:

1. Seznamte se s nástroji, které umožňují převádět zachycený zvuk při hře na piano do notového zápisu. Zvolte takovou implementaci systému, která umožňuje své začlenění do externích aplikací.
2. Navrhněte systém, který by pro zvolené klavírní cvičení pořízené jako audionahrávka umožnil kontrolovat, zdali přehrávání tohoto cvičení odpovídá zápisu.
3. Realizujte vámi navržený systém jako mobilní nebo desktopovou aplikaci.
4. Testujte vhodnost vámi vytvořené aplikace na několika lidech, kteří jsou ve hře na piano různě pokročilí a zpracujte jejich hodnocení této aplikace.

Literatura:

- Beauchamp, J. W.: Analysis, synthesis, and perception of musical sounds. Springer. 2007
- Russel, S., Norvig, P.: Artificial Intelligence, A Modern Approach, Pearson, 2009

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

Vedoucí práce:	**Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1. listopadu 2020
Datum odevzdání:	19. května 2021
Datum schválení:	12. listopadu 2020

Abstrakt

Účelem této práce je navrhnout a implementovat aplikaci pro cvičení hry na klavíru. Hlavní nevýhodou již existujících aplikací je omezený výběr skladeb, které lze cvičit, protože tyto skladby jsou pevně zapsány v paměti. Aplikace která je výsledkem této práce, vyřeší daný problém – uživatel bude moci nahrát do aplikace libovolnou klavírní nahrávku, kterou chce procvičovat a aplikace se postará o vytvoření procesu cvičení. Proces cvičení spočívá v ukázce určitým způsobem upravených not uživateli a současně aplikace pomocí mikrofonu kontroluje, co uživatel hraje. Největší výzvou dané diplomové práce bylo nalézt způsob, jak přesně a rychle klasifikovat audio signál z mikrofonu. Daná úloha byla vyřešena pomocí dvou nezávislých neuronových sítí s různou architekturou, které byly trénovány na různých datových sadách. Za účelem odůvodnění zvoleného řešení budou uvedeny všechny potřebné teoretické muzikální a vědecké pojmy a metody, které mají k tomuto tématu vztah. Výsledná aplikace bude testována dle třech parametrů: přesnost, rychlost a uživatelská použitelnost.

Abstract

The purpose of this work is to design and implement an application for piano practice. The main disadvantage of existing applications is the limited selection of songs that can be practiced. The application, which is the result of this work, will solve the problem – the user will be able to upload to the application any piano record he wants to practice, and the application will take care of creating a training process. The training process consists of showing the user a certain way of editing notes and simultaneously controlling what the user is playing with a microphone. The biggest challenge of the diploma thesis was to find a way to accurately and quickly classify the audio signal from the microphone. This problem was solved using two independent neural networks with different architectures, which were trained on different data sets. To justify the chosen solution, all the necessary theoretical musical and scientific concepts and methods that are directly related to it will be presented. The resulting application will be tested in next respects: accuracy, speed, noise resistance and the usability of the user.

Klíčová slova

strojové učení, neuronové sítě, automatizovaná transkripce hudby

Keywords

machine learning, neural networks, automatic music transcription

Citace

USTINOV, Nikita. *Nástroj pro podporu cvičení klavírních skladeb*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Dr. Ing. František Zbořil, Ph.D.

Nástroj pro podporu cvičení klavírních skladeb

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Dr. Ing. Zbořila, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Nikita Ustinov
18. května 2021

Poděkování

Děkuji vedoucímu práce Dr. Ing. Františeku Zbořilu, Ph. D. za metodickou a odbornou pomoc při zpracování této diplomové práce.

Výpočtové prostředky byly poskytnuty v rámci projektu „e-Infrastruktura CZ“ (e-INFRA LM2018140) poskytovaného v rámci programu Projekty velkých infrastruktur výzkumu, vývoje a inovací.

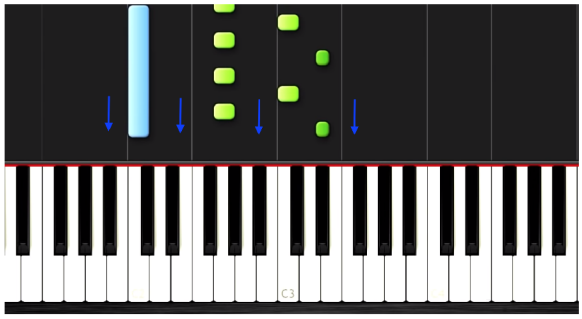
Obsah

Kapitola 1

Úvod

V dnešní době existuje velké množství aplikací pro cvičení hry téměř pro jakýkoliv hudební nástroj, včetně klavíru. Hlavní nevýhodou těchto aplikací je omezený výběr skladeb, které lze cvičit, protože tyto skladby jsou pevně uloženy v paměti a zpravidla je nelze přidat externě. Účelem této práce je navrhnout a implementovat aplikaci pro cvičení hry na klavíru, kde uživatel může procvičovat jakoukoliv skladbu, k níž má nahrávku. Bodové shrnutí interakcí uživatele a aplikace je následující:

- Uživatel nahraje do aplikace audio záznam klavírní hudby.
- Aplikace převede daný záznam do MIDI, nebo jiného vnitřního formátu.
- Aplikace bude následně: ukazovat uživateli noty které jsou reprezentovány barevnými obdélníky a na základě příjmu z mikrofonu bude kontrolovat, co uživatel hraje na klavíru. Obdélníky postupně padají na příslušné klávesy v grafickém uživatelském rozhraní (dál GUI). Když se obdélník dotkne klávesy, program zastaví padající obdélníky a bude čekat, dokud uživatel nezmáčkne příslušnou klávesu na klavíru. Po zmáčknutí bude padání obdélníku pokračovat. Obr. 1.1 demonstruje možné GUI.
- Na konci cvičení se uživateli zobrazí celkový čas, který aplikace čekala na zmáčknutí jakékoliv klávesy. Tato hodnota bude působit jako skóre, které se uživatel musí snažit vylepšit.



Obrázek 1.1: Možný GUI [35].

Tento cvičící proces může být velmi užitečný pro uživatele, který nezná muzikální teorii, noty a notový zápis. Grafická prezentace not ve tvaru padajících obdélníků může uživatele naučit, kdy a jakou klávesy musí zmáčknout, aby zazněla správná melodie.

Zjednodušením oproti standardní úloze automatizované transkripce hudby (automatic music transcription, AMT [13]) je skutečnost, že v dané aplikace nebudeme brát v úvahu polyfonickou hudbu, tj. hudbu která se skládá z více zdrojů (hlas, kytara, klavír). Nadále budeme předpokládat, že na vstupu bude nahrávka obsahující pouze zvuk klavíru. Návrh takové aplikace představuje poměrně komplikovanou úlohu, zejména kvůli následujícím dvěma problémům. První výzvou je přesný překlad zvukového záznamu uživatele do vnitřní reprezentace (např. MIDI formát), na základě které bude uživatel cvičit. Druhým problémem je pořízení zvuku z mikrofonu a v reálném čase kontrola zvuku, který uživatel hraje. Aplikace kontroluje, zda zvuk odpovídá originální nahrávce, aby včas zastavila padání not (obdélníků). Hlavním rozdílem mezi prvním a druhým problémem je požadavek na přesnost a rychlost. V prvním případě je důležitá přesnost transkripce hudby do not, zatímco rychlost zpracování není důležitá. Ve druhém případě, při kontrole hry uživatele v reálném čase, je doba zpracování jednoho zvukového fragmentu na prvním místě. Oba problémy lze redukovat na problém klasifikace přehrávaných not podle fragmentu zvukového záznamu. Nejběžnější způsoby klasifikace zvuku v současné době jsou:

- Neuronové sítě [19].
- Nezáporná maticová faktorizace [14].
- Skryté Markovské řetězce [39].
- Techniky zpracování signálu (např. autokorelace [47]).

V kapitole 2.3 budou výše uvedené techniky blíže popsány. V kapitole 2.4 budou uvedeny nejvýznamnější práce založené na metodách zmíněných v kapitole 2.3. Na konci kapitoly 2.4 bude odůvodněno použití neuronových sítí jako jádra naší práce.

Naše řešení je založeno na použití dvou různě velkých neuronových sítí (každá pro jednotlivé části programu) s různými architekturami. První neuronová síť bude zaměřena na přesnou transkripci nahrávky do notového zápisu. Tato síť se všemi na ni kladenými požadavky již existuje a byla zveřejněna v práci [23]. Danou síť jednoduše integrujeme do výsledné aplikace. Druhá síť bude vytvořena vlastními silami a použita za účelem průběžné kontroly zvuku z mikrofonu během cvičícího procesu. Velká část práce se bude zabývat vytvořením vlastní v reálném čase fungující neuronové sítě: výběru architektury, formátu vstupních dat, přípravou datasetu a samotnému trénování. Přínos této práce můžeme shrnout následně:

- Nalezení vhodné architektury pro klasifikaci zvuku v reálném čase pomocí kombinace několika state-of-the-art řešení.
- Úprava dat pro cvičení tak, aby bylo možné vycvičit neuronovou síť v kratší době (odstranění nevyváženosti v datech a různých chyb).
- Úprava architektury tak, aby neuronová síť samostatně dokázala predikovat přesnost vlastní klasifikace (pomocí doplňujícího výstupu, který bude natrénován na chybách hlavního výstupu).

Celkové řešení bude testováno, zda odpovídá časovým požadavkům na zpracování v reálném čase, a zda je použitelné pro konečného uživatele.

Kapitola 2

Teoretický základ

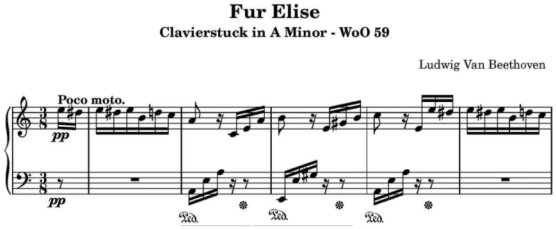
Daná kapitola vymezuje pojmy a principy, které jsou nezbytné pro pochopení dané práce.

2.1 Zvuk a reprezentace zvuku

Zvuk je fyzický jev, který se šíří ve formě elastických vln, mechanických vibrací v prostoru. Stejně, jako libovolná vlna, můžeme zvuk popsat amplitudou a frekvencí. Člověk dokáže vnímat zvuku s frekvencí od 16—20 Hz do 15—20 kHz. Klavír produkuje zvuk od 27.5-4186 Hz (odpovídá notám A0 až C8, viz dále)[17].

2.1.1 Tón, nota

Tón je stálý zvuk s určitou frekvencí[1]. Z kombinace tónů se skládá hudba. Notou nazýváme grafickou reprezentaci tónu, viz Obr. 2.1. Pro pojmenovaní not používáme následující symboly: *C, D, E, F, G, A, B*. Pokud je frekvenční poměr dvou zvuků stupněm dvojky (2^x), poté tyto zvuky vnímáme jako podobné a značíme stejným symbolem. Interval mezi těmito zvuky se nazývá oktáva. Každá oktáva se dále dělí na 12 intervalů, které se nazývají půltóny. Pro rozlišení mezi stejnými notami různých oktáv se používají číslice, např. *C1, D4* atd.[17] Síla tónu udává jeho hlasitost.



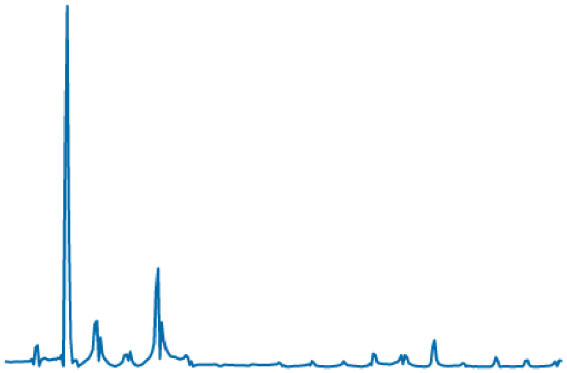
Obrázek 2.1: Hudební zápis no

2.1.2 Harmonická řada

Značným problémem pro úlohu AMT je skutečnost, že pokud zazní jakýkoliv tón, neslyšíme pouze frekvenci příslušnou danému tónu (základní frekvenci nebo $F0$). Navíc slyšíme i

[1] V literatuře se často uvádí, že tón je stálý zvuk s určitou výškou. Výšku můžeme odvodit jednoduchými výpočty z frekvence. Proto v textu odvozujeme výšku od frekvence.

harmonické frekvence, které jsou odvozeny od $F0$. Daný typ frekvence se nazývá harmonická řada. Proto je na Obr. 2.2 mmísto jednoho vrcholu odpovídajícího tónu E vidět několik vrcholů, jež odpovídají základní frekvenci a následně harmonickým odvozeným frekvencím [29, 38].

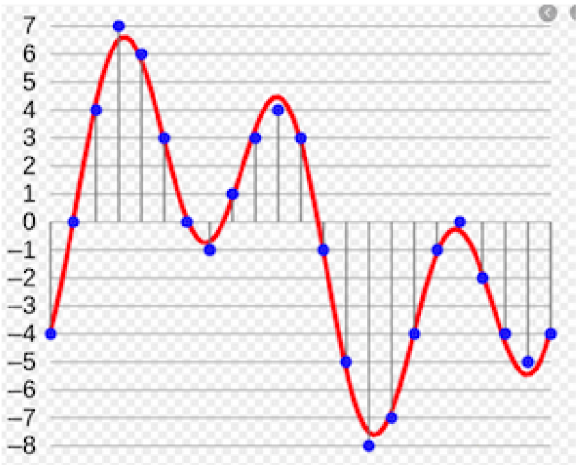


Obrázek 2.2: Spektrum jednoho tónu E (kytara).

2.1.3 Formáty audio

Chceme-li zvuk zapsat nebo přehrát na elektronickém zařízení, je potřeba použít vhodný zvukový formát, který moderní zařízení dokáží zpracovat.

WAV. Základním formátem, kterému rozumí všechny počítače s operačním systémem Windows, je *WAV* formát. Ve své podstatě WAV představuje kontejner (obálku) určitého audio formátu, typicky formátu *PCM* (Pulse Code Modulation). PCM je poměrně starý formát, ale i v něm lze zapsat velmi kvalitní zvuk. PCM soubor obsahuje sekvenci vzorků v čase, kde každý vzorek udává amplitudu (sílu) signálu v určitý okamžik (viz Obr. 2.3). Nevýhodou daného formátu je jeho příliš velká velikost. Právě tento problém se snažily vyřešit pozdější formáty, například MP3 [27].



Obrázek 2.3: PCM ukázka

MIDI. Musical Instrument Digital Interface (MIDI) je jedním z nejvýznamnějších formátů přenosu hudby. Převážně se používá pro ukládání kompozic produkovaných různými

hudebními nástroji. Na rozdíl od WAV se MIDI nesnaží uložit každý vzorek v každý okamžik, ale ukládá pouze noty (klávesy, struny apod.), které byly spuštěny v daný okamžik, čímž dokáže silně redukovat velikost cílového souboru [45]. Na Obr. 2.4 lze vidět obsah standardního MIDI souboru.

```
<message note_on channel=0 note=52 velocity=40 time=1521>
<message note_on channel=0 note=64 velocity=42 time=0>
<message note_on channel=0 note=59 velocity=43 time=211>
<message note_on channel=0 note=64 velocity=0 time=141>
<message note_on channel=0 note=64 velocity=44 time=0>
<message note_on channel=0 note=55 velocity=45 time=14>
<message note_on channel=0 note=59 velocity=0 time=126>
<message note_on channel=0 note=59 velocity=46 time=0>
<message note_on channel=0 note=52 velocity=0 time=141>
```

Obrázek 2.4: MIDI soubor je strukturován jako seznam událostí. Každá událost obsahuje značku, kód klávesy (note), čas (time) a rychlost (velocity). Kód klávesy je speciální MIDI kód, který jednoznačně odkazuje na klávesu na hudebním nástroji. Čas události uvádí, kolik časových jednotek uplynulo od předchozí události. Rychlost udává, s jakou silou byla zmáčknuta klávesa. Pokud se rychlost rovná nule, tato událost odkazuje na skutečnost, že klávesa byla právě uvolněna.

2.1.4 Frekvenční reprezentace zvuku

Těžko analyzovat zvuk pouze na základě amplitud roztažených v čase. Většinou chceme vědět jaké frekvence obsahuje zvuková nahrávka. Proto budeme potřebovat transformovat amplitudy (PCM vzorky) do frekvencí které tento zvuk obsahuje, jinými slovy budeme potřebovat spektrum vstupního signálu.

DFT. Za účelem získání spekra vstupního signálu můžeme použít diskrétní Fourierovou transformaci (DFT), která se vypočítá podle definice 1. Porovnání dvou záznamů zvuku je vidět na Obr. 2.5.

Definice 1. Nechť $\{x_n\}_{0 \leq n \leq N}$ je vzorkovaný audio signál. k-ta složka jeho frekvenčního spektra, X_k, se počítá podle následujícího vzorce [15]:

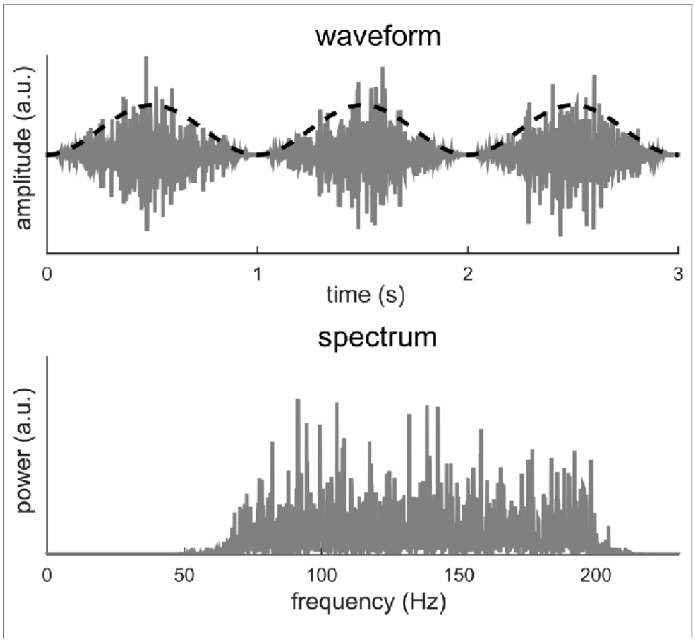
$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{i2\pi}{N} kn}.$$

DFT vezme na vstupu vzorkovaný signál v čase a na výstupu budeme mít stejný signál rozložený ve frekvenční doméně. V praxi se však používá jeho zrychlená podoba FFT (Fast Fourier Transformation), která časově náročnější $\mathcal{O}(N \log N)$ oproti $\mathcal{O}(N^2)$ u DFT.

Zvuková reprezentace pomocí lineárně uspořádaných frekvencí neodpovídá přesně tomu, jak zvuk vnímá člověk, proto není v praxi klasické spektrum příliš populární.

Mel-spektrum. Jednou z možných alternativ DFT je *Mel-spektrum*. Daný způsob vytváření spektra je inspirován přírodou, tedy způsobem, jak člověk dokáže vnímat různé frekvence[2]. Toto spektrum je komprimováno ve frekvenční ose, a proto může být efektiv-

[2]Člověk lépe rozlišuje nízké frekvence. Se zvýšením frekvence rozlišovací schopnost člověka logaritmicky klesá.



Obrázek 2.5: Příklad transformace signálu pomocí DFT: amplitudy roztažené v čase (nahoře) a odpovídající spektrum (dole). [53].

nější ve velikosti při zachování nejdůležitějších informací.[40] Mel transformace můžeme popsat pomocí tohoto vzorce:

$$m = 2595 \log_{10} \left(1 + \frac{f}{700}\right),$$

kde f je transformovaná frekvence [10].

CQT. Další možnou alternativou DFT je *Constant-Q (CQT)* spektrum. CQT spektrum poskytuje frekvenční reprezentaci s logaritmickými měřítky středních frekvencí, což je dobře sladěno s frekvenčním rozložením tónu, proto se CQT používá převážně tam, kde by základní frekvence not měly být přesně identifikovány, například pro rozpoznávání akordů, nebo transkripci not. Je však nutné zdůraznit, že výpočet CQT je náročnější než DFT.[13] Transformaci klasického spektra do CQT spektra popisuje vzorec:

$$f_c(k_{lf}) = f_{min} \times 2^{\frac{k_{lf}}{\beta}}, \tag{2.1}$$

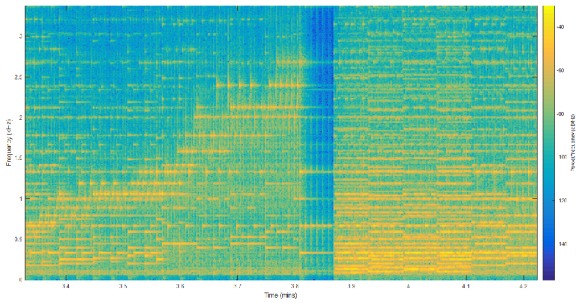
kde f_{min} je minimální analyzovaná frekvence, k_{lf} - index filtru, β - počet bínů v oktávě.

Cepstrum. Další reprezentací zvuku je tzv. *cepstrum* [8]. Jedná se o výsledek inverzní Fourierovy transformace (IDFT) na spektru s logaritmickým měřítkem. Cepstrum je popsáno vzorcem 2.2. Daná transformace se často používá za účelem zkoumání zvukového signálu (typicky lidské řeči).

$$c(m) = \mathcal{F}^{-1}[\ln|\mathcal{F}s(n)|^2] \tag{2.2}$$

Okénkové funkce. Výše uvedené transformace mají jednu negativní vlastnost: pokud ji provedeme – ztratíme informaci o čase. Například při transformaci hudební skladby pomocí DFT budeme znát frekvenci a amplitudy z dané skladby, ale nebudeme vědět, v jakém

okamžiku se tyto frekvence objevily. Proto pokud chceme signál (zvuk) analyzovat v čase, potřebujeme jinou reprezentaci, kterou je spektrogram. Spektrogram dostaneme tak, že vstupní signál rozdělíme na rámce (vynásobením signálu posuvným oknem) a v každém rámci spočítáme DFT. Příklad spektrogramu lze vidět na Obr. 2.6 [26].



Obrázek 2.6: Spektrogram klavírní skladby [10].

Při rozdělení signálu na rámce je hlavním problémem skutečnost, že přesnost v časové a ve frekvenční doméně se navzájem vylučují. Pokud zvyšujeme rozlišení v čase (např. při použití malého okna), ztratíme přesnost frekvenčního rozlišení, tj. bude těžší rozlišit od sebe dvě podobné frekvence. A naopak, pokud zvýšíme rozlišení (použitím velkého okna), dokážeme snadno odlišit podobné frekvence, ale nebudeme mít jistotu, v jakém okamžiku daná frekvence zazněla. DFT spektrogram s malým a velkým posuvným oknem je zobrazen na obrázku 2.8. Toto je jeden z několika problémů, který omezuje možnost získat správné výsledky v AMT úlohách [26].

Dalším problémem je zvolení tvaru okna, kterým vynásobíme signál. Toto je důležité, protože při využití této operaci dochází ke změně spektra původního signálu. Zvolením různých tvarů oken se snažíme co nejvíce redukovat danou změnu. Nejčastějšími tvary pro posuvné okna jsou [54]:

- Pravoúhlé okno – nemění původní signál::

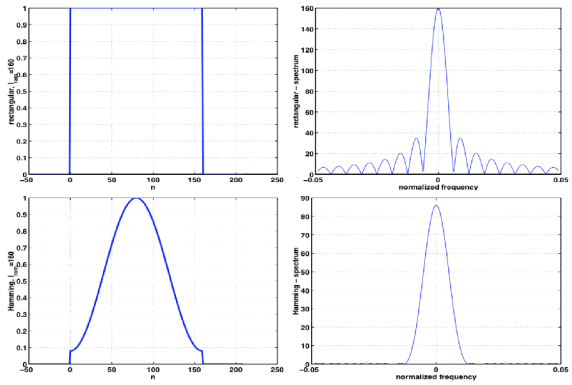
$$w[n] = \begin{cases} 1 & \text{pro } 0 \leq l_{ram} - 1, \\ 0 & \text{jinde.} \end{cases}$$

- Hammingovo okno – utlumí signál na okrajích:

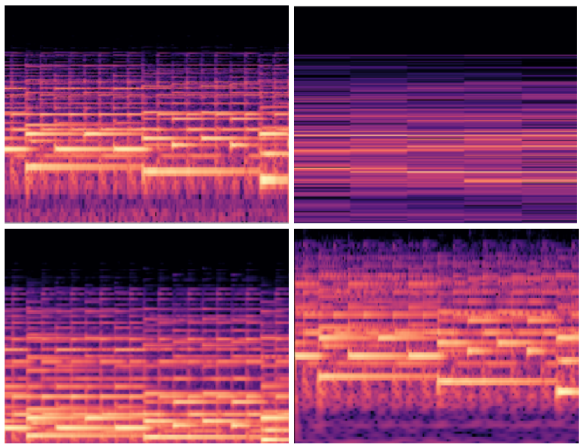
$$w[n] = \begin{cases} 0.54 - 0.46 \cos \frac{2\pi n}{l_{ram}-1} & \text{pro } 0 \leq l_{ram} - 1, \\ 0 & \text{jinde.} \end{cases}$$

Příklad využití různých oken a výsledné spektrum je znázorněno na Obr. 2.7

Jak lze vidět, pravoúhlé okno dokáže produkovat užší šipku okolo potřebné frekvence, ale zároveň toto okno kolem dané šipky způsobuje velké artefakty. Hammingovo okno má naopak poněkud širší šipku, ale artefakty ve výsledném spektru téměř zanedbatelné. Příklad využití různých transformací na stejném signál je znázorněn na Obr. 2.8.



Obrázek 2.7: Srovnání pravoúhlého (vlevo nahoře) a Hammingova (vlevo dole) okna a výsledků jejich aplikací (vpravo).



Obrázek 2.8: Porovnání různých transformací. DFT spektrogram s krátkým posuvným oknem (nahoře vlevo), DFT spektrogram s velkým posuvným oknem (nahoře vpravo), Mel spektrogram (dole vlevo), CQT spektrogram (dole vpravo).

2.2 Existující nástroje

Tato kapitola popisuje nejvýznamnější nástroje či aplikace pro automatickou transkripci hudby a cvičení hráče. Na základě analýzy těchto nástrojů bude v praktické části vytvořen koncept budoucí aplikace.

2.2.1 AMT nástroje

Onsets and Frames. Jedná se o ukázku výsledků práce [23]. Automatická transkripce funguje na základě webového prohlížeče. Uživatel nahraje audio a systém zobrazí příslušné MIDI. Tento nástroj neobsahuje žádné další nástroje pro úpravu výsledného MIDI. Cena – zdarma [37].

Transcribe. Tento nástroj lze zároveň zařadit do dvou skupin: AMT a nástroj pro cvičení hry. AMT část je dostatečně klasická – uživatel poskytne audio, program vyprodukuje MIDI, které je graficky znázorněno v GUI. V případě zájmu uživatel toto MIDI upraví a uloží. Cvičící část je poměrně jednoduchá. Uživateli je nabídnuto zvolit fragment pro cvi-

čení. Program bude graficky ukazovat, jaké klávesy musí uživatel mačkat v jaký okamžik. Fragment se bude opakovat, dokud uživatel bude chtít pokračovat.

Jedná se o nativní program, který podporuje Windows, Linux a MacOS. Má 30 dnů zkušební verzi, poté požádá jednorázovou platbu ve výši 39$ [6].

MusicTrans. MusicTrans je nativní AMT software s podporou platformy Windows, MacOS, Linux a Android. Přesnost transkripce není uvedena. Obsahuje několik nástrojů pro korekci výsledné transkripce člověkem (zpomalení/opakování fragmentů atd.). Má 5 dnů užívání zdarma. Poté lze zdarma provádět transkripci pouze několik minut od kompozice. Plná licence stojí 249Kč [4].

MelodyScanner. Je transkripční systém který funguje na základě prohlížeče. U pianových nahrávek deklaruje přesnost 86% a dokáže zpracovat i jiné hudební nástroje (avšak bez uvedené přesnosti). Obsahuje podporu transkripcí videa z YouTube. Cena licence - 5.99$ za měsic [3].

2.2.2 Nástroje pro cvičení hry

Flowkey: Learn piano. Jedná se o populární cvičící aplikaci. Aplikace ukazuje zároveň noty v klasickém pojetí a nahrávku, jak ji hraje člověk. Aplikace kontroluje proces cvičení pomocí kabelového připojení k syntezátoru nebo pomocí mikrofonu. Podle hodnocení na webu můžeme říci, že kontrola hry přes mikrofon funguje průměrně. [1]

Simply Piano. Alternativní aplikace k Flowkey s jedním rozdílem. Daná aplikace se orientuje při kontrole hry uživatele pouze dle mikrofonu [2].

Synthesia. Synthesia je aplikací, účelem které je naučit hru na klavír. Od předchozích aplikací se liší ve dvou věcech. Program kontroluje hru uživatele pomocí senzoru na zařízení, tj. není zde možnost cvičit jinak než na zařízení, na kterém je aplikace instalována. Dalším rozdílem je způsob, jakým uživatel pochopí, kterou klávesy má zmáčknout. Displej je rozdělen na dvě části, v dolní části je klavír s klávesami, na nichž uživatel může hrát. V horní části je prostor, na kterém se značným předstihem objevovují obdélníky a postupně padají směrem dolů. V okamžiku, kdy se obdélník dotkne klávesy v dolní části zařízení, uživatel musí zmáčknout příslušnou klávesu. Jinak se padání obdélníků zastaví a aplikace bude čekat, než se uživatel dotkne správné klávesy [5]. Příklad GUI můžeme vidět v Příloze 7.1.

2.3 Modely pro transkripci zvuku

Pro účely klasifikace, nebo rozpoznávání tónů můžeme použít velké spektrum přístupů. Tato kapitola stručně popisuje nejvýznamnější techniky.

2.3.1 Tradiční přístupy

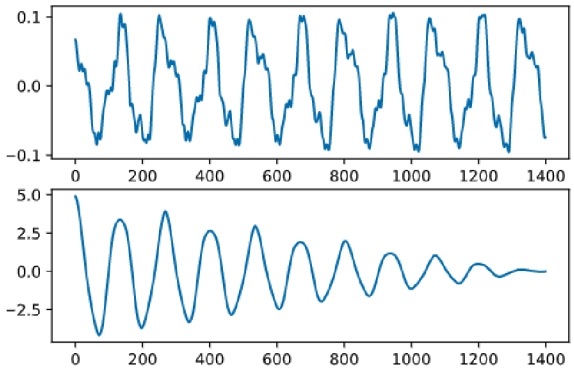
Tradiční přístupy jsou techniky, kdy člověk přímo určuje algoritmus, pomocí kterého bude výsledný program fungovat. Tyto přístupy můžeme rozdělit na detekci tónu ve frekvenční a časové oblasti.

Autokorelace. Typickým příkladem detekce tónů v časové oblasti je autokorelace. Matematicky danou operaci vyjadřuje vzorec 2.3. Slovně autokorelaci můžeme popsat jako operaci, která provádí porovnání signálu se zpožděnou kopii daného signálu. Delka zpoždění se nazývá *lag*. Jakmile budou zkoumány všechny možné lagy, bude nalezen lag, při kterém

byl signál nejvíce podobný. Daný lag představuje hledaný tón (F0). Výsledek této operace můžeme vidět na Obr. 2.9.

$$r_x[l] = \sum_{n=0}^{n=N-1-l} x[n]x[n+l] \qquad l = 0, 1, ..., N-1, \tag{2.3}$$

kde l je zkoumaný lag [54].



Obrázek 2.9: Příklad autokorelace: původní signál v amplitudách (nahoře) a odpovídající autokorelace daného signálu (dole).

Cesptrální analýza. Pro detekci tónů ve frekvenční oblasti můžeme použít cepstrum signálu. Za F0 se považuje frekvence, která má nejvyšší cepstrální koeficient v rozmezí povolených koeficientů.

Tradiční přístupy mají velmi dobrou schopnosti zobecnění a fungují stejně dobře na všech hudebních nástroje [14]. Nevýhodou dané skupiny přístupů je skutečnost, že většinou fungují dobře pouze pro detekci jednoho tónu. Pokud je potřeba detekovat tónů více, mají dané metody často s detekcí problémy. Metody popsány níže v této kapitole jsou vhodnější pro detekci několika tónů najednou.

2.3.2 Nezáporná maticová faktorizace (NMF)

NMF je skupinou algoritmů, cílem které je rozložit matici $V \in \mathbb{R}_{\geq 0}^{M \times N}$ na dvě nezáporné matice $D \in \mathbb{R}_{\geq 0}^{M \times K}$ a $A \in \mathbb{R}_{\geq 0}^{K \times N}$, tak aby $V = D \times A$. Nezápornost matic D a A poskytuje velmi přesné omezení. Výsledné matice D a A jsou tedy mnohem jednodušší pro následující analýzu, než původní matice V. V kontextu AMT úlohy NMF můžeme představit tak, že matice V je původní spektrogram, kde n-ty sloupec je spektrum v okamžiku n. Poté je matice D slovník, který reprezentuje rozložení energie vůči každému tónu. Matice A je aktivační matice, která ukazuje, jak aktivně zazněl každý tón z D v čase [14].

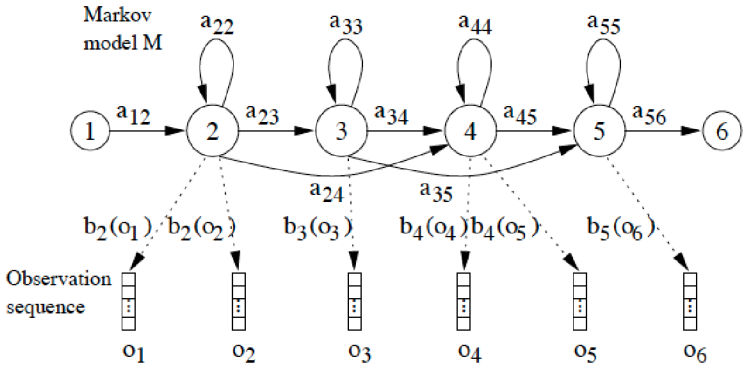
Nalezení matic D a A je optimalizačním $NP - hard$ problémem. Proto se hledá pouze přibližné přípustné řešení. Typický algoritmus se v několika iteracích snaží zmenšit vzdálenost mezi $D \times A$ a V. Nejjednodušším příkladem algoritmů je multiplikativní metoda aktualizace váhy od autorů Daniel D. Lee a H. Sebastian Seung [30].

2.3.3 Skryté Markovovy modely

Definice 2. Nechť X_n a Y_n jsou stochastické procesy v diskrétním čase a $n \geq 1$. Potom dvojice (X_n, Y_n) je *Skrytý Markovův Model (HMM)* pokud platí:

- X_n je Markovův proces a není přímo pozorovatelný (skrytý)
- $\boldsymbol{P}(Y_n \in A \mid X_1 = x_1, \ldots, X_n = x_n) = \boldsymbol{P}(Y_n \in A \mid X_n = x_n)$, pro každé $n \geq 1, x1, \ldots, x_n$, a měřitelnou množinu A.

Jinými slovy, HMM je statistický model, který simuluje činnost procesu podobného Markovovu procesu s neznámými parametry. Při učení je potřeba najít neznámé parametry na základě pozorovatelných. Získané parametry mohou být použity v další analýze, například pro rozpoznávání vzorů [28]. Pokud existuje klasifikační úkol pouze s jednou třídou, může HMM vypadat následovně:



Obrázek 2.10: Skrytý Markovův model, který dokáže říci, zda vzorek patří pouze do jedné konkrétní třídy [28].

Jak lze vidět, HMM se skládá z několika skrytých stavů. Každý stav má uvnitř Gaussovou funkci. Přechod mezi stavy se provádí s určitou pravděpodobností (na Obr. 2.10 - a_{ij}). Na vstup (do stavu 1) se podává určitá sekvence. Výstupem poté bude míra podobnosti vstupní sekvence a sekvence, které HMM se naučil. Tato míra podobnosti se určuje tak, že každý stav podle Gaussovy funkce počítá pravděpodobnost, kdy i-ty element vstupní sekvence je shodný s j-tým elementem naučených sekvencí. Na základě těchto dílčích pravděpodobností výstupního stavu poté spočítá celkovou pravděpodobnost, že vstupní sekvence je shodná se naučenými sekvencemi. Při výpočtu může nastat problém, kdy má vstupní sekvence jiný počet elementů než počet stavů HMM(I <> J). Tento problém můžeme řešit sestavením různých cest pomocí Dynamic Time Warping metody (popsána v kapitole 2.5.3). Pokud je potřeba rozpoznávat více tříd, (což je typické) přidáme k tomuto modelu stejné modely, jež se snaží naučit jiné vzorky. Vstupní vzorek je vzorkem třídy, jejíž model má na výstupu nejvyšší pravděpodobnost.

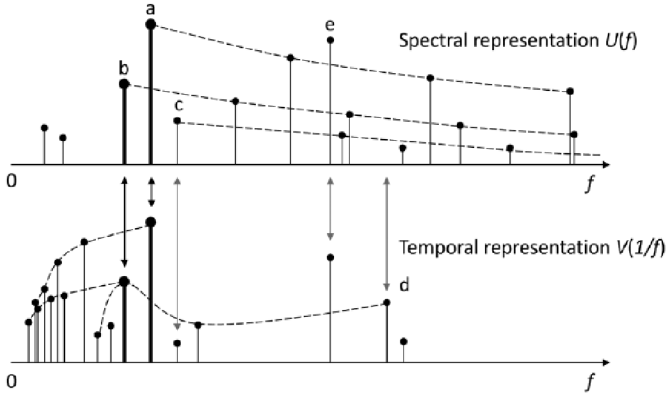
2.3.4 Neuronové sítě

Umělá neuronová síť (dále „neuronová síť“) je matematický model a jeho softwarová, nebo hardwarová implementace je postavena na principu organizace a fungování biologických neuronových sítí – sítí nervových buněk živého organismu. Tato koncepce vznikla při studiu procesů probíhajících v mozku a při snaze simulovat tyto procesy [19]. Neuronové sítě se skládají z neuronů, které jsou navzájem propojeny do určité architektury. Architektury se typicky rozděluje na několik vrstev: vstupní, skrytá a výstupní. Pokud se skrytá vrstva skládá z několika vrstev, tato neuronová síť se nazývá hluboká.

2.4 Existující přístupy pro transkripci zvuku

Než začneme navrhovat vlastní aplikaci, je potřeba prozkoumat již existující řešení. To nám umožní odhalit slepé uličky a přímo posouvat správně zvolenou technologii dopředu. V této kapitole jsou uvedeny různé způsoby klasifikací audio vzorků klavírní hudby, což je klíčovým krokem k transkripci audiosignálu do not.

Autoři [31] zároveň analyzovali log-scaled spektrum a cepstrum za účelem lepšího výběru kandidátních not v určité části kompozice. Obr. 2.11.



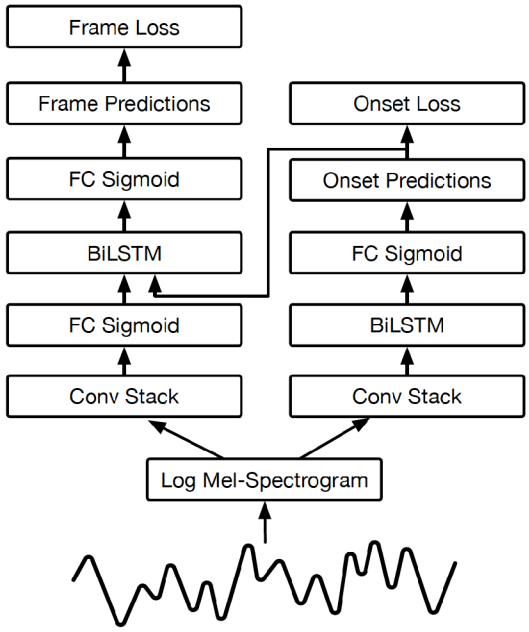
Obrázek 2.11: Log-scaled spektrum a cepstrum [31].

Následně se uplatnily klasické metody zpracování signálu. Jako výsledek ukazuje daný přístup lepší hodnoty přesnosti pouze na jednom datasetu ze sedmi. Avšak stále se jedná o dobrý alternativní pokus bez použití neuronových sítí.

V [25] autoři použili HMM (viz definice 2) pro klasifikaci audio vzorku za účelem odhalení plagiátu v nově vymyšlených kompozicích. Navržený model dosahuje průměrné přesnosti přes všechny typy datasetů ve výši 67 procent, což je relativně málo. Takto malá přesnost je způsobena příliš malým datasetem, kterým autoři disponovali.

Práce [22] používá neuronové sítě a je založena na myšlence, že některé rámce jsou důležitější než ostatní. Tyto rámce jsou rámce s začátkem noty, protože po začátku znění noty energie začne klesat. A právě v těchto rámcích máme nejvíce informací pro klasifikaci vzorků. Za účelem nalezení těchto rámců byl nacvičen zvláštní detektor začátku noty založený na konvolučních a rekurentních vrstvách. Vstupem pro tento detektor je Mel-spektrogram. Jehož výstupem je 88 pravděpodobnost (klavír má 88 kláves), že určitá nota právě zazněla v tomto rámci. Tento výstup je přímo použit jako doplňkový vstup pro detektor not. Celková architektura sítě je zobrazena na Obr. 2.12.

Metoda navržená v [23] je založena na metodě představené v [22]. Neuronová síť (model) byla převzata pouze s drobnými úpravami, které se týkaly pouze zvýšení počtu neuronů ve vrstvách. Závažným rozdílem [23] od [22] je to, že autoři v [23] použili pro cvičení modelu jiný dataset, čímž dosáhli obrovského nárůstu přesnosti. Vzniklý dataset se jmenuje MAESTRO v1. Byl sestaven pomocí nahrávek vytvořených během devíti let pianových konkurzů. Obsahuje dvojice synchronizované na 3 ms (audio – MIDI). Takto přesná synchronizace byla dosažena minimalizací vzdáleností mezi CQT rámcem z audia a vygenerovaným audiem na základě příslušného MIDI. Minimalizace vzdálenosti byla provedena pomocí algoritmu DTW (více v kapitole 2.5.3). V Tab. 2.1 můžeme vidět porovnání různých datasetů s MAESTRO v1.



Obrázek 2.12: Architektura sítě [22]

Dataset	Počet skladeb	Celková délka	Počet not(milion)
SMD	50	4.7	0.15
MusicNet	60	15.3	0.58
MAPS	208	17.9	0.62
MAESTRO v1	1184	172.3	6.18
MAESTRO v2	1282	201.2	7.13

Tabulka 2.1: Porovnání datasetů [23].

Lze vidět, ze MAESTRO v1 má téměř desetkrát větší počet not oproti MAPS datasetu (na němž byl cvičen model z prací [22]).

2.5 Učení neuronových sítí

I když jsou ostatní přístupy k AMT z kapitoly 2.3 dobré, při porovnání s neuronovými sítěmi jsou velmi pozadu (na základě kapitoly 2.4). Proto bude naše řešení založeno převážně na neuronových sítích.

Učením neuronových sítí se označuje proces, během kterého se určují váhy (typické) každého neuronu při procházení cvičné sady. Na začátku učení jsou váhy inicializovány malými čísly kolem 0. Při zpracování každého vzorku se váhy sítě mírně mění. Epochou se rozumí počet iterací trénování, pokud neuronová síť zpracovává každý vstup z datové sady právě jednou. Trénovací proces typicky končí, když přesnost neuronové sítě dosáhne předem nastavené konstanty anebo se přesnost přestane zlepšovat po určitém počtu epoch. Můžeme rozlišit několik přístupů k učení neuronových sítí:

Učení s učitelem. Je druh učení, který je typický pro problémy jako regrese a klasifikace. Při učení s učitelem trénovací sada obsahuje dvojici (x, y) kde x je vstup pro neuronovou síť a y je požadovaný výstup neuronové síti. Po přijetí na vstupu x, síť produkuje vlastní výstup $y\prime$. Při porovnání $y\prime$ a y trénovací proces dostává informaci, v jakém směru musí upravovat váhy každého neuronu. Porovnání $y\prime$ a y a extrakce informace o směru a velikosti chyb se provádí pomocí funkce ztrát (více kapitola 2.5.1) [44].

Učení bez učitele. Učení bez učitele se používá obvykle pro úlohy klasterizace, kdy nevíme přesný počet tříd. Na rozdíl od učení s učitelem datová sada pro daný druh úlohy neobsahuje složku. To znamená, že síť nebude dostávat ke každému vstupu správnou odpověď. Učení poté spočívá v nalezení zákonnosti (souvislosti) mezi vstupními vzorky a seskupení dat do různých skupin [52].

Učení s podpořením. Daný typ učení se používá, pokud existuje model, který rozhoduje o akci v situovaném prostředí na základě neuronové sítě. Problém je v tom, že model nemá žádné znalosti o systému. Model však může provádět experimenty — určitým způsobem ovlivňovat prostředí a sbírat odezvy[3] od prostředí. Právě na základě těchto dat se neuronová síť snaží naučit udělat správnou předpověď – volit nejlepší následující akci tak, aby v dlouhodobé perspektivě model dosáhl předem stanoveného cíle (stavu v systému). Tento typ učení se obvykle používá pro naučení správné strategie umělého agenta (robota, hráče ve hře atd.) [12].

Transfer learning. Jedná se o metodu, kterou nelze zařadit do výše uvedených kategorií. Tato metoda může být aplikována ke každé z nich. Transfer learning se používá, pokud nemáme dostatečné množství dat nebo výpočetních zdrojů na trénování vlastní neuronové sítě. Daná metoda spočívá v použití již naučené neuronové sítě na jiných datech. Z této sítě odstraníme poslední vrstvy a začneme síť trénovat na vlastní datové sadě. Pokud byla síť učena na velké datové sadě, obsahuje všechny potřebné nízko úrovňové vzorce, které je potřeba extrahovat ze vstupu. Nám zbývá naučit poslední vrstvy tak, aby byla síť schopna použít tyto vzorce pro námi zadaný specifický úkol.

2.5.1 Ztrátová funkce

Funkce ztrát je srdcem algoritmů strojového učení. Používají se pro extrakci informací o chybě výstupu neuronové sítě. Na základě této chyby optimalizační algoritmus dokáže změnit váhy tak, aby se síť zlepšila. Existuje mnoho druhů ztrátových funkcí, které jsou účinné pouze pro malý seznam úloh. Proto je výběr ztrátové funkce velmi důležitý.

Obecně je můžeme rozdělit na dvě skupiny podle druhu úloh, na kterých se uplatňuje model (neuronová síť):

- Regresní
- Klasifikační

Tato diplomová práce je zaměřena na klasifikaci vstupní nahrávky. Zde je obtížné použít modely, které jsou přímo zaměřeny na regresní úlohy. Proto za účelem dodržení zadání diplomové práce budou dále popsány pouze aktivační funkce, které jsou důležité pro klasifikační úlohy.

[3] Odezvou se zde rozumí nový stav prostředí.

Mean Square Error (MSE). je klasickou funkcí ztrát ve strojovém učení. Níže můžeme vidět matematický vzorec.

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

kde n je počet výstupních neuronů.

Nevýhodou MSE je, že druhá mocnina v $(y_i - \hat{y})^2$ silněji penalizuje velké odchylky, což zhoršuje rychlost konvergence. MSE se původně používalo pro všechny typy úloh, ale s časem bylo zjištěno, že pro klasifikační úlohy existují lepší alternativy. Nyní se MSE a ostatní podobné ztrátové funkce (Mean Bias Error, Mean Absolute Error) používají v regresních úlohách.

Binary Cross Entropy (BCE). je možnou alternativou k MSE [34]. Obvykle se používá pro úlohy binární klasifikace. Můžeme ji popsat následujícím vzorcem.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \times \log(p(y_i)) + (1 - y_i) \times \log(1 - p(y_i)).$$

BCE se snaží přiblížit rozložení hodnot výstupních neuronů penalizací za chybný a za příliš pochybující výstup. Používá se v klasifikačních úlohách, kde vstupní vektor můžeme zatřídit pouze do jedné třídy [33].

Co když je potřeba zařadit vstupní vektor do více tříd? Poté se jedná o multi-label klasifikační úlohu. V takovém případě se používá ***Categorical Cross Entropy***. Tato ztrátová funkce je popsána následujícím vzorcem.

$$L(X_i, Y_i) = -\sum_{j=1}^{N} y_{ij} \times log(p_{ij}),$$

kde Y_i je one-hot vektor[4] p_{ij} je pravděpodobnost, že vzorek i patří do třídy j.

2.5.2 Metriky

Abychom pochopili, zda se neuronová síť zlepšuje, či nikoliv, musíme zavést určité metriky, které nám budou tuto informaci poskytovat. Těchto metrik existuje celá řada. Nelze zvolit pouze jednu, protože různé metriky poskytují různé vlastnosti kvality neuronové sítě. V dalším textu budou popsány nejdůležitější metriky pro učení neuronových sítí [42].

Nejprve je potřeba vysvětlit pojem *chybové matici*. Chybová matice je specifická tabulka, která umožňuje vizualizovat výkon algoritmu (typicky se jedná o učení s učitelem). Každý řádek matice představuje instance v predikované třídě, zatímco každý sloupec představuje instance v reálné třídě (nebo naopak). Příklad chybové matice pro dvě třídy můžeme vidět v tabulce 2.2. Z ní dostáváme čtyři důležité hodnoty: True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN). Tyto hodnoty budou použity pro výpočet potřebných metrik.

Nejpopulárnější metrika je *accuracy*. Jedná se o poměr správných odpovědí vůči celkovému počtu odpovědí. Vypočítá se podle vzorce 2.4:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.4}$$

[4]One-hot vektor, je vektor čísel 0,1. Délka vektoru se rovná počtu tříd a každé číslo uvádí, zda příslušný vzorek patří do určité třídy [34].

	y = 1	y = 0
y'= 1	True Positive	False Positive
y'= 0	False Negative	True Negative

Tabulka 2.2: Chybová matice pro dvě třídy, kde: y' – je predikce modelu, y – je skutečná hodnota reálného světa.

Tato metrika je standardem, ale je téměř neužitečná při existenci nevyváženosti v trénovacích datech. Je to dáno tím, že accuracy je silně závislá na poměru počtu prvků v každé třídě.

Na rozdíl od accuracy není *precision* a *recall* závislé na poměru počtu prvků v třídách, a proto je můžeme použít i pro nevyvážená trénovací data. Můžeme je vypočítat podle vzorců 2.5 a 2.6.

$$precision = \frac{TP}{TP + FP}, \tag{2.5}$$

$$recall = \frac{TP}{TP + FN}. \tag{2.6}$$

V některých případech je potřeba popsat kvalitu neuronové sítě jedinou hodnotou. K tomu se používá metrika *f1*. Jedná se o harmonický průměr precision a recall. Výpočet této metriky je popsán vzorcem 2.7.

$$f1 = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall}, \tag{2.7}$$

kde β - váha důležitosti precision a recall. Tato metrika se rovná jedničce, pokud se precision a recall budou rovnat jedničce. f1 bude blízko nule, pokud je precision nebo recall blízko nule.

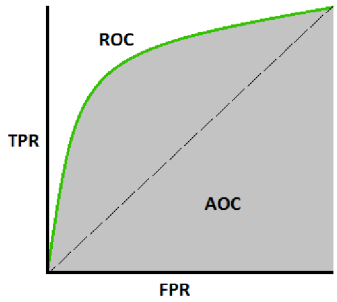
Dále je nutné zmínit *ROC(Receiver Operating Characteristics) křívku*. Ta se používá v případě, když potřebujeme určitým způsobem ohodnotit klasifikační model, který produkuje spojené hodnoty v určitém intervalu. U daného typu klasifikačního modelu bude výstup klasifikace záležet na prahu, který zvolíme. Například se jedná o klasifikační neuronovou síť, která říká, jestli je na obrázku kočka, či ne. Tato neuronová síť bude produkovat hodnoty v intervalu [0, 1]. 1 bude znamenat, že na obrázku je kočka, 0 – že tam nikdo není. Problém nastává v okamžiku, když neuronová síť pro určitý vstup vyprodukuje výstup 0.3. Abchychom určili, kam zařadit danou klasifikaci, potřebujeme stanovit práh (např. 0.5). Poté budeme jakýkoliv výstup sítě větší než daný práh klasifikovat jako přítomnost kočky a všechny výstupy menší, nebo rovnající se prahu, jako nepřítomnost kočky. A nyní bude hodnocení dané neuronové sítě bude záležet na zvoleném prahu. K ohodnocení takových modelů se používá ROC křivka. Jedná se o křivku, která je definována v prostoru $R \in (TPR, FPR)$, kde *TPR a FPR* jsou dány následujícími rovnicemi [7]:

$$True\ positive\ rate = \frac{TP}{(TP + FN)}, \tag{2.8}$$

$$False\ positive\ rate = \frac{FP}{(TN + FP)}, \tag{2.9}$$

Plocha pod *ROC* křivkou se nazývá *AUC(Area Under The Curve)*. Vizualizace daných metrik je uvedena na Obr. 2.13.

Také se často sleduje metrika *Loss*. Jedná se o hodnotu ztrátové funkce, která byla určena pro danou neuronovou síť (více 2.5.1.).



Obrázek 2.13: ROC a AUC metriky [7].

2.5.3 Příprava trénovací sady

Modely strojového učení jsou typicky velmi závislé na velikosti a kvalitě trénovací sady (dále „dataset“). Na velikosti závisí, jak dobře bude model rozumět reálným datům. Dle kvality datasetu závisí, jak rychle model a trénovací algoritmus dokáží extrahovat užitečné znalosti z dat a převést je do parametru modelu.

I presto ze pořízení velikého datasetu je problém není moc triviální. V rámci diplomové práce je tento problém těžko ovlivnitelný. Na nás však závisí kvalita dat, které použijeme do modelu (například neuronové sítě). Proto bude následující sekce věnovaná zlepšení kvality datasetu.

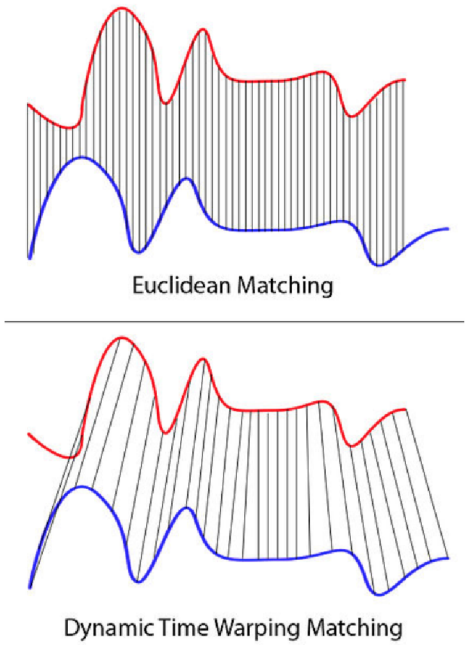
V případě, že se jedná o učení s učitelem, je potřeba, aby každému vstupu odpovídal správný výstup. V případě práce se zvukem (dataset má tvar dvojic – audio, MIDI) je potřeba mít správně zarovnán dataset: část audia musí obsahovat pouze noty, které jsou uvedené v seznamu MIDI. Zarovnání můžeme provést pomocí DTW (vic 2.5.3).

Je nezbytné mít v datech určitou úroveň šumu, aby model mohl po cvičení porozumět skutečným datům. Avšak úroveň šumu nesmí být pro model matoucí.

Pokud známe empirické údaje o datech, je dobrým řešením převést data do jiné representace, která zachovává i tuto znalost. Tím docílíme velké redukce času, který je potřeba věnovat trénování modelu, protože model již nebude nucen se naučit tuto znalost. V případě, že se jedná o audio nahrávku, a chceme-li tuto audio nahrávku klasifikovat, nemusíme modelu poskytnout PCM vzorky (víz 2.3). Model lépe dokáže klasifikovat nahrávku, pokud dostane do vstupu místo PCM vzorků frekvenční reprezentaci této nahrávky. Algoritmy strojového učení jsou velmi závislé na rozsahu hodnot vstupních dat. Proto je potřeba v datasetu předem změnit měřítko (normalizovat). Rozsah hodnot a nová škála je závislá na typu použitého modelu (typu aktivační funkce v případě neuronových sítí). Například pro SVM klasifikátor nejlépe vyhovuje rozsah vstupních hodnot [-1,1] a směrodatná odchylka 0.5 s centrem v 0 [20].

Dynamic Time Warping (DTW). Je algoritmus pro nalezení míry podoby mezi dvěma sekvencemi s různou délkou. Hlavní problém s porovnáním těchto sekvencí je, že jejich porovnání eukleidovskou vzdáleností často selhává. To je dáno tím, že eukleidovská vzdálenost se vypočítá postupným porovnáním prvků se stejnými indexy. Následně v situaci, kdy budou porovnány dvě téměř stejné sekvence (vektory), ale jedna z nich bude posunuta o jeden prvek, eukleidovská vzdálenost může považovat tyto vektory za úplně odlišné. Na rozdíl od eukleidovské vzdálenosti se DTW snaží co nejlépe namapovat prvky (indexy) jednoho vektoru na druhý. Poté se za nejlepší mapování považuje délka (rozdíl) mezi vektory [18].

Rozdíl mezi eukleidovskou metrikou a DTW je uveden na Obr. 2.14.



Obrázek 2.14: Rozdíl při porovnání pomocí euklidovské metriky (nahoře) a DTW (dole) [18].

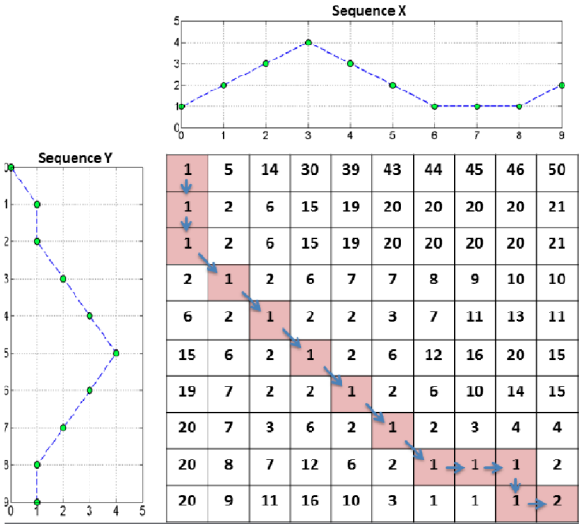
Nechť máme dva vektory r a t s délkami k a n, potom zarovnání probíhá následujícím způsobem [18]:

1. Sestavena matice M, kde každý prvek $M(i, j)$ je dán výpočtem vzdálenosti mezi vektory $r(i)$ i $t(j)$.
2. Probíhá výpočet cesty mezi prvkem $M(0, 0)$ a $M(k, n)$.
3. Buňky matice, které patří k určité cestě, se sčítají. Tím dostáváme kumulovanou vzdálenost dvou vektorů.
4. Tato kumulovaná vzdálenost poté může být normalizována.
5. Nejmenší vzdálenost je považována za vzdálenost mezi vektory r a t.
6. Průběh výpočtu můžeme vidět na obrázku dole.

Grafické znázornění porovnání dvou sekvencí pomocí DTW je uvedeno na Obr. 2.15.

2.6 Architektury neuronových sítí

Tato kapitola se zabývá kompletním popisem složení neuronových sítí, tedy od jednoduchého neuronu, přes jednotlivé vrstvy, až do state-of-the-art architektur, které mají význam pro tuto práci.



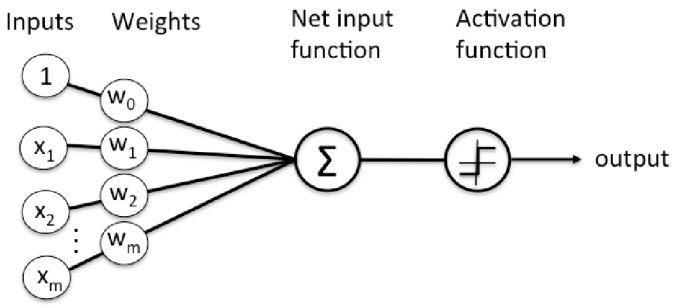
Obrázek 2.15: Výpočet DTW [36].)

2.6.1 Neuron

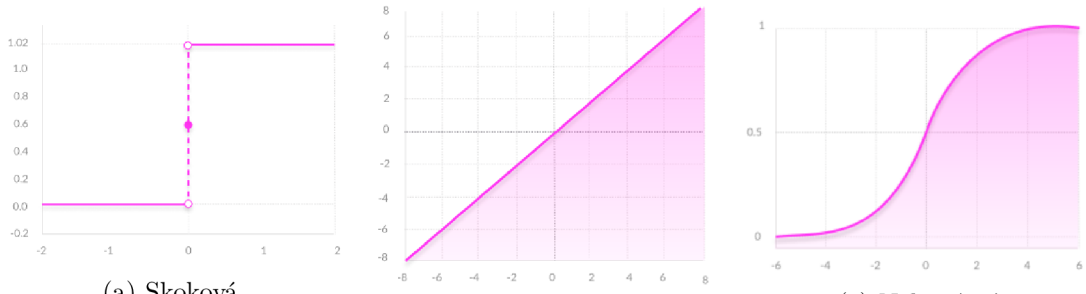
Umělý neuron je umělý neuronový síťový uzel, což je zjednodušený model přírodního neuronu. Matematicky je umělý neuron obvykle představován jako nelineární funkce jediného argumentu – lineární kombinace všech vstupních signálů. Tato funkce se obvykle nazývá *aktivační funkce*. Jinými slovy se na začátek spočítá u (vzorec 2.10). Poté se předává na vstup určité aktivační funkci $f()$.

$$u = \sum_{i=0}^{n} w_i x_i, \tag{2.10}$$

kde n je počet neuronů, ke výstupem kterých se připojen daný neuron. Výsledek je odeslán na jeden výstup. Takové umělé neurony jsou kombinovány v sítích - spojují výstupy některých neuronů se vstupy ostatních [19]. Na Obr. 2.16 lze vidět model umělého neuronu – Preceptron. Jedná se o nejjednodušší verzi umělého neuronu. V praxi se typicky používají složitější modely.



Obrázek 2.16: Model umělého neuronu [46]



(a) Skoková

(b) Lineární

(c) Nelineární

Obrázek 2.17: Průběh různych aktivačních funkcí [11].

2.6.2 Aktivační funkce

Funkce aktivace neuronové sítě jsou klíčovou součástí hlubokého učení, které se vyvinulo paralelně s architekturou neuronové sítě. Aktivační funkce určují výstup modelu hlubokého učení, jeho přesnost, a také výpočetní efektivitu trénování modelu. Aktivační funkce mají také hlavní vliv na schopnost neuronové sítě konvergovat a rychlost konvergence[5], nebo v některých případech mohou aktivační funkce zabránit neuronovým sítím v konvergování [11].

Aktivační funkce mohou být:

- Skokové (binární):

$$y = \begin{cases} 0 & \text{pro u} < \theta, \\ 1 & \text{pro u} \geq \theta. \end{cases}$$

- Lineární:

$$y = ku \tag{2.11}$$

- Nelineární:

$$y = tanh(u) \tag{2.12}$$

Grafický průběh různých typů aktivačních funkcí je znázorněn na Obr. 2.17.

V současnosti se převážně používá nelineární funkce, protože umožňují vytvářet komplexní mapování mezi vstupy a výstupy sítě, které jsou nezbytné pro učení a modelování komplexních dat, jako jsou obrázky, video, audio a datové soubory, jež jsou nelineární a zároveň mají vysokou dimenzi [9].

V současné době existuje velké množství různých aktivačních funkcí. Některé z nich jsou univerzální a mohou být použity na všech vrstvách sítě. Některé funkce jsou specifické a používají se pouze za určitých podmínek. Dále budou popsány nejdůležitější aktivační funkce, jež se používají v klasifikačních úlohách.

Sigmoida (obrázek vpravo 2.17) je klasickou aktivační funkcí, která byla dříve velmi populární a je vhodná pro všechny vrstvy. Má rozsah hodnot od (0, 1) a centrum v 0.5. Matematicky je popsaná vzorcem 2.13.

$$y = \frac{1}{1 + e^{-z}}, \tag{2.13}$$

[5]Konvergencí se zde rozumí přibližování ke správnému řešení.

Hyperbolický tangens (dále tanh) je další v minulosti populární aktivační funkce. Má rozsah hodnot od (-1, 1) a centrum v 0. Kvůli tomu má lepší rychlost konvergenci než sigmoida. Matematicky je popsaná vzorcem 2.14.

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{2.14}$$

Nevýhodou tanh a sigmoidy je poměr — (výpočetní složitost/přínos). Samostatně jsou tyto funkce velmi užitečné, ale nyní existují více efektivní alternativy. Navíc tyto funkce jsou velmi citlivé na problém mizejícího gradientu[6].

Průlomem se stal vynález funkce ReLU (Rectified Linear Unit). Matematicky je tato funkce popsána vzorcem 2.15.

$$y = max(0, u). \tag{2.15}$$

Kvůli jednoduchosti je učení neuronu, které mají uvnitř ReLU, několikrát rychlejší než učení tanh nebo sigmoidy. Ale ReLU neurony mají problém, pokud mají na vstupu 0. V tomto případě klasický ReLU neuron „zemře“ (jeho výstupem bude od tohoto okamžiku vždy 0). To způsobí snížení rozlišovací schopnosti sítě. Můžeme problém řešit pomocí drobných modifikací, například Leaky ReLU.

Leaky ReLU — je modifikací ReLu funkce„ která je popsána následujícím vzorcem.

$$y = \begin{cases} u & \text{pro } \geq 0, \\ \alpha & \text{pro } < 0, \text{ kde } \alpha << 1. \end{cases}$$

Pokud Leaky ReLU dostane jako vstup zápornou, nebo nulovou hodnotu, na rozdíl od klasického ReLU vrátí malou zápornou hodnotu. Tímto zabraní „smrti“.

ELU (Exponential Linear Unit) — je silnou alternativou Leaky ReLU. ELU můžeme popsat následujícím vzorcem.

$$y = \begin{cases} u & \text{pro } u \geq 0, \\ \alpha(e^u - 1) & \text{pro } z \leq 0, \end{cases}$$

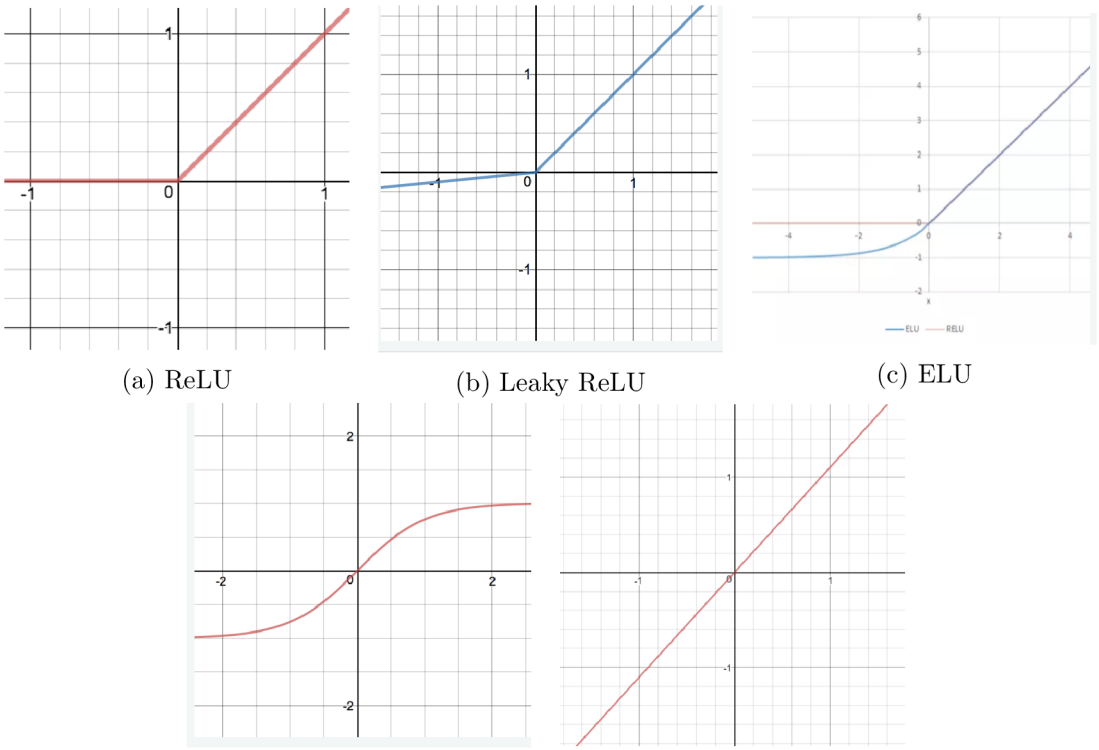
kde α je předem stanovená konstanta. Daná aktivační funkce také umí produkovat záporné hodnoty. Nevýhodou je však pro vstupní hodnoty > 0 může „vybuchnout“ (vygenerovat na výstupu příliš velké hodnoty).

Softmax — jedná se o specifickou aktivační funkci, která je přímo určena pro výstupní vrstvu. Má rozsah výstupních hodnot (0, 1), přitom se suma výstupních hodnot všech výstupních neuronů rovná 1. To znamená, že síť, která má na výstupu Softmax aktivační funkci, může specifikovat, na kolik procent je jistota, že na vstupu vidí určitou třídu. Proto je nyní Softmax standardem pro klasifikační úlohy. Níže je matematicky popsán výstup neuronu s indexem j.

$$y_j = \frac{e^{u_j}}{\sum_{k=1}^{m} e^{u_k}},$$

kde m je počet výstupních neuronů [32]. GGrafické znázornění průběhu aktivačních funkcí lze vidět na Obr. 2.18.

[6]Mizející gradient problém — velmi důležitý problém při učení neuronových sítí. Při zvětšení počtu skrytých vrstev rychle klesá schopnost prvních vrstev se učit. To je dáno specifikem učení neuronových sítí a konkrétně BGD (Backpropagation Gradient Descent) algoritmem (https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484).



(a) ReLU (b) Leaky ReLU (c) ELU

(d) Tanh (e) Linear

Obrázek 2.18: Průběh aktivačních funkcí.

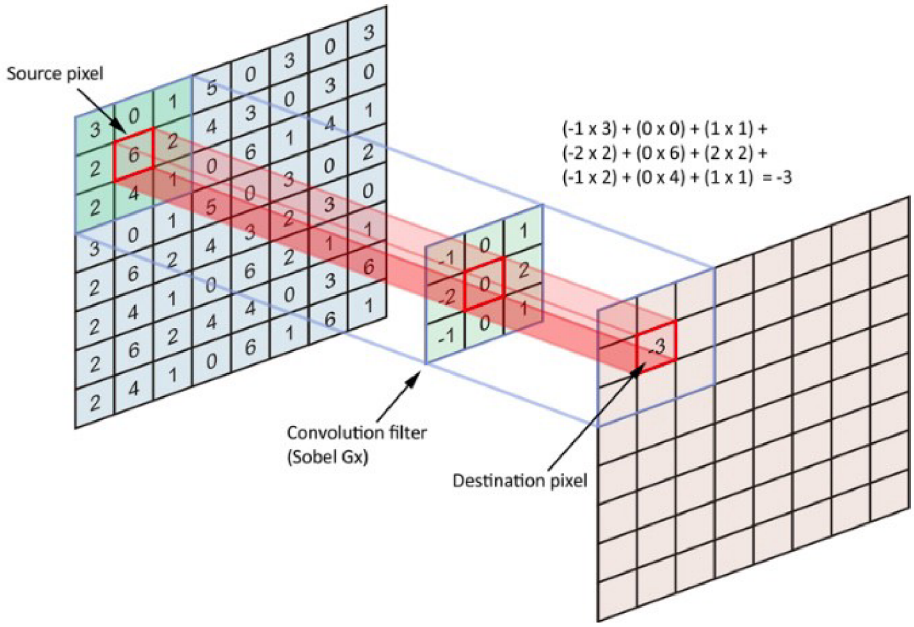
2.6.3 Vrstvy neuronových sítí

Jak bylo uvedeno výše, vrstvy představují skupinu podobných neuronů. Některé vrstvy jsou univerzální – můžeme je použít téměř v každé architektuře, některé jsou specifické – používají se pouze ve specifických architekturách. Tato kapitola popisuje nejčastější vrstvy neuronových sítí, které se používají v architekturách důležitých pro klasifikační úlohy.

Dense (plně propojená) vrstva. Klasická vrstva, kde je každý neuron propojen s každým výstupním neuronem předchozí vrstvy [19]. Typicky jsou to poslední vrstvy neuronových sítí. Tato vrstva silně zvětšuje počet trénovacích vah (větší trénovací doba), ale je velmi užitečná pro sumarizaci dílčích příznaků předchozích vrstev.

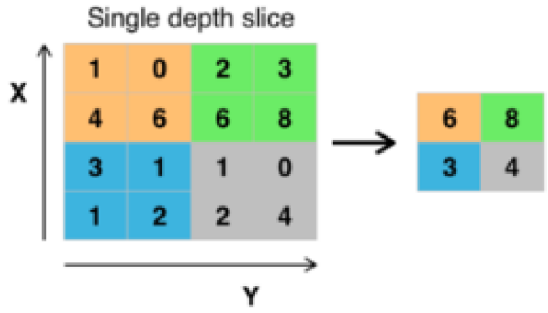
Konvoluční vrstva. Konvoluční vrstva představuje množinu neuronů, které jsou odlišné od klasického chápání neuronu [49]. Na rozdíl od Dense vrstvy je neuron v konvoluční vrstvě připojen (¨vidí¨) pouze k malému lokálnímu poli z předchozí vrstvy a během výpočtu prochází svým lokálním oknem celou předchozí vrstvu. Konvoluční neuron je reprezentován konvolučním jádrem (maticí). Při učení se síť snaží naučit správné hodnoty tuto matici. Pro získání výstupu neuronu se jeho konvoluční jádro pohybuje nad vstupním obrázkem (nebo nad výstupem předchozí vrstvy) a vypočítá lineární kombinaci hodnot pod jádrem, kde jako koeficienty lineární kombinace používá hodnoty v jádře. Na Obr. 2.19 můžeme vidět ukázku procházení konvolučního jádra.

Tento typ vrstvy zpravidla reaguje na předem naučenou vlastnost: čáru, hranu, kruh. Na základě získaných vlastností dokáží plně propojené vrstvy snadno klasifikovat obrázek.



Obrázek 2.19: Výpočet výstupu konvolučního neuronu pro jednu polohu [49].

Sdružující vrstva. Také známa jako pooling vrstva, je nelineární komprese mapy rysů. Skupina pixelů (obvykle ve velikosti 2 × 2) je zkomprimovaná na jeden pixel a podléhá nelineární transformaci. Nejčastěji se používá funkce max (). Transformace ovlivňují nesouvislé obdélníky nebo čtverce, z nichž každý je stlačen do jednoho pixelu a je vybrán pixel s maximální hodnotou. Operace sdružování výrazně snižuje prostorový objem obrazu. Shromáždění se interpretuje následovně: pokud již byly některé známky zjištěny v předchozí operaci konvoluce, pak tento podrobný obrázek již není zapotřebí pro další zpracování a je zkomprimován do méně podrobného. Kromě tohoto filtrování již nepotřebných částí pomáhá zabránit přeučení neuronové sítě. Sdružující vrstva se obvykle vkládá za konvoluční vrstvu před další konvoluční vrstvu. Na Obr. 2.20 je vidět výpočet pooling vrstvy pro jeden vstup.



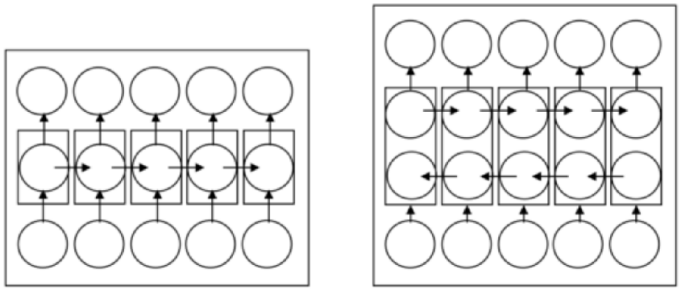
Obrázek 2.20: Komprimace 16 výstupů předchozí vrstvy do 4 [49].

Kromě sdružování s maximální funkcí můžeme použít i další funkce – například průměrnou hodnotu, nebo určitou normalizaci.

LSTM. LSTM je vrstva, která se skládá z LSTM modulů. Modul LSTM je modul pro cyklické sítě, který může ukládat hodnoty pro krátkou i dlouhou dobu. Na vstupu bere vzorky, které uspořádá v čase v jednom směru (t1, t2, t3 atd.). Závazné je, že modul LSTM nepoužívá aktivační funkci uvnitř svých rekurzivních komponent. A tímto způsobem rekurentní neuronové sítě (více dále) používající LSTM jednotky částečně řeší problém mizejícího gradientu, protože LSTM jednotky umožňují gradientu průtok beze změny. Sítě LSTM však stále mohou trpět problémem explodujícího gradientu [41].

BiLSTM. BiLSTM je podobná LSTM s tím rozdílem, že BiLSTM při generaci výstupu (odpovědi) bude brát v potaz vzorky uspořádané ve dvou směrech v čase (t1, t2, t3 a zároveň t3, t2, t1) [41].

Schematické zobrazení LSTM a BiLSTM sítí lze vidět na Obr. 2.21.



Obrázek 2.21: Schematické znázornění architektury LSTM (vlevo) a BiLSTM (vpravo) vrstev [41].

Batch normaliation. Batch normaliation (dal BN) je vrstva (nebo technika) která poskytuje možnost regularizaci sítě. Myšlenka spočívá v tom, že každá vrstva přijímá jako vstup výstup předchozí vrstvy, ve které může být prakticky jakýkoli tenzor, jehož hodnoty jsou údajně nějak rozloženy. Ale pro každou vrstvu je nejlepší situace, když do ní vstupují tenzory s hodnotami z pevné distribuce, pak by nebylo nutné hledat transformaci invariantní k parametrům distribuce vstupních dat. Toto vykonává BN – normalizuje výstupy předchozí vrstvy podle vzorce 2.16 přičemž parametry normalizace (μ a σ) se mění během trénovacího procesu [43].

$$y = \frac{x - \bar{x}}{\sigma} \tag{2.16}$$

kde x - je vstup dané vrstvy, y - výstup.

Avšak tento BN má i nevýhody, zejména:

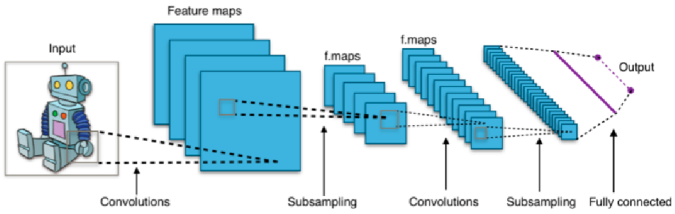
- Aby BN fungoval, je potřeba použít relativně velký batch size.
- BN je výpočetně náročnou vrstvou.
- BN může produkovat artefakty. Příklad je uveden v příloze 7.2.

Kvůli těmto nevýhodám se v posledních letech mnoho výzkumných skupin snaží odstranit BN vrstvu bez ztráty výhod, které BN poskytuje [51].

2.6.4 Architektury neuronových sítí

Existuje řada architektur, které jsou zaměřeny na různé účely. Následující kapitola popisuje nejdůležitější architektury neuronových sítí vztažených ke klasifikaci signálu (včetně audia).

Konvoluční neuronové sítě (CNN). CNN jsou speciální architektura umělých neuronových sítí, navržená Janem Lekunem v roce 1988, a zaměřená na efektivní rozpoznávání vzorů. V roce 2012 tento typ sítě zvítězil v soutěži o klasifikaci obrázků s velkým náskokem oproti konkurenci. Od této doby to byl standard v klasifikaci obrázků. Myšlenka konvolučních neuronových sítí je tedy střídat konvoluční vrstvy a sdružující vrstvy. Struktura sítě je jednosměrná (bez zpětné vazby). Klasickou strukturu neuronové sítě můžeme vidět na Obr. 2.22.



Obrázek 2.22: Typická architektura konvoluční neuronové sítě [49]

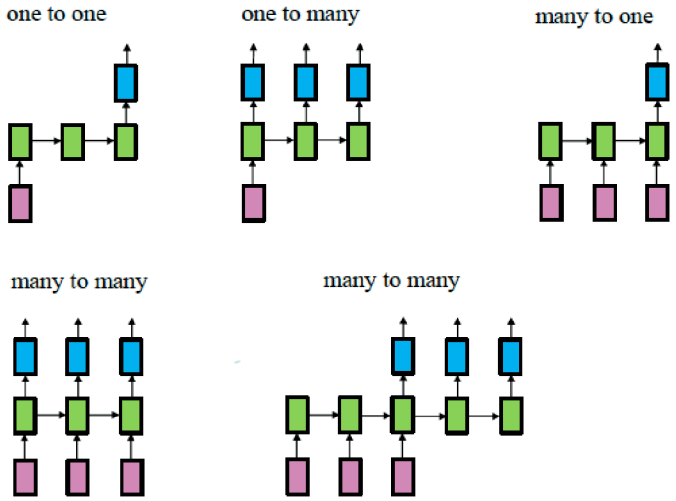
Rekurentní neuronové sítě (RNN). RNN jsou typ neuronových sítí, které jsou primárně určeny k práci s časově závislými sekvencemi (řeč, písmo, hudba). Na rozdíl od vícevrstvých acyklických neuronových sítí mohou RNN používat svou vnitřní paměť ke zpracování sekvencí libovolné délky. Pro rekurentní sítě, od jednoduchých po komplexní, bylo navrženo mnoho různých architektonických řešení. V poslední době je nejrozšířenější síť s dlouhodobou a krátkodobou pamětí (LSTM) [41].
Obecně můžeme rekurentní sítě rozdělit na [41]:

- One to one – používá se pro predikci a extrapolaci.
- One to many – často se navazuje na konvoluční síť a používá se pro automatický popis obrázku.
- Many to one – je vhodná pro klasifikaci sentimentů.
- Many to many (bez posuvu) – používá se pro strojový překlad.
- Many to many (s posuvem) – lze aplikovat pro klasifikaci videa na úrovni snímků.

Na Obr. 2.23 je uveden přehled druhů rekurentních neuronových sítí.

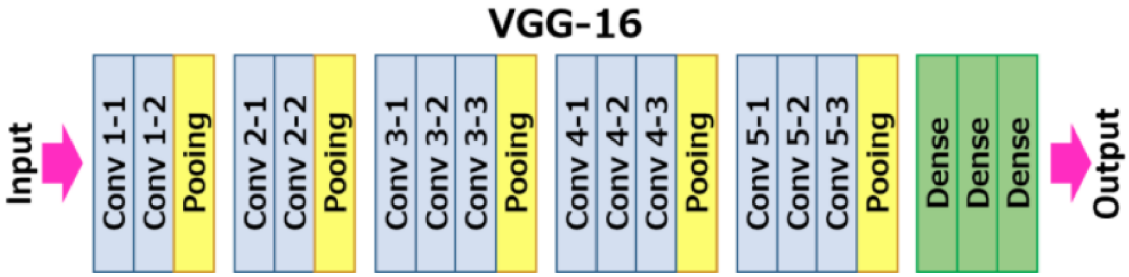
Klasická RNN má však určité problémy. Teoreticky může sledovat libovolné dlouhodobé závislosti ve vstupních sekvencích, avšak problém je v tom, že pokud trénujeme klasickou RNN pomocí zpětného šíření chyby, gradienty, jež se šíří zpět, mohou „zmizet“ (mohou konvergovat k nule) nebo „explodovat“ (konvergovat k nekonečnu) [41].

Důležité reprezentanty. V této části jsou popsány důležité reprezentanty konvolučních neuronových sítí. To jsou nejmenší a nejrychlejší neuronové sítě mezi konvolučními neuronovými sítěmi. Přesnost dále uvedených sítí je stále dobrá, a proto jsou začleněny do výčtu v této práci.



Obrázek 2.23: Schematické rozdělení RNN [41].

VGG16. VGG16 je konvoluční neuronová síť navržená K. Simonyanem et al. ([21]), což je vylepšená verze AlexNet[7]. Na rozdíl od AlexNet VGG16 nahrazuje velké konvoluce (velikosti 11 a 5 v první a druhé konvoluční vrstvě) několika malými konvolucemi o velikosti 3x3, jedna po druhé. Počet trénovacích parametrů se však zvýšil na 138 milionů. Při testování na ImageNet[8] VGG16 dosahuje přesnosti 92,7 procenta[9]. VGG16 můžeme vidět na Obr. 2.24.



Obrázek 2.24: Architektura VGG16 kde: Conv – konvoluční vrstva, Pooling – sdružující vrstva a Dense – plně propojená vrstva.

MobileNetV2. MobileNetV2 je konvoluční neuronová síť od Google, která je náhradou MobileNetV1. Tyto sítě mají malou výpočetní složitost (pouze 3,4 milionu trénovacích parametrů), a proto se primárně používají na mobilních zařízeních [48]. Architekturu dané sítě můžeme vidět na Obr. 2.26

[7]AlexNet je konvoluční neuronová síť. Byla představena na konkurzu pro rozpoznávaní objektů v obraze v roce 2012. Od tohoto momentu se konvoluční sítě považují za standart v klasifikaci obrazů.

[8]mageNet je dataset v úloze rozpoznávání objektů v obraze. ImageNet se skládá z více než 14 milionů obrázků patřících do 1 000 tříd

[9]Síť VGG16 byla trénována několik týdnů pomocí grafických karet NVIDIA TITAN BLACK. Přitom se měření přesnosti na ImageNet provádí tak, že výsledek je počítán jako správný, pokud neuronová síť zařadí správnou odpověď mezi pěti nejpravděpodobnějšími objekty na obrázku.

V MobileNetV2 se používají nové typy vrstev:

- Relu6
- Hloubkově oddělitelná konvoluce

Relu6. *Relu6* je drobnou modifikací klasické ReLu funkce. Jediným rozdílem je, že maximální výstupní hodnota Relu6 může být 6. Nový matematický zápis této funkce je uveden níže.

$$f = min(max(0, u), 6)$$

Relu6 pomáhá zachovat robustnost sítě při využití výpočtu s nízkou přesností.

Hloubkově oddělitelná konvoluce (Depthwise Separable Convolution). Je nová (relativně komplikovaná) modifikace konvoluční vrstvy. Hlavní myšlenky této vrstvy jsou:

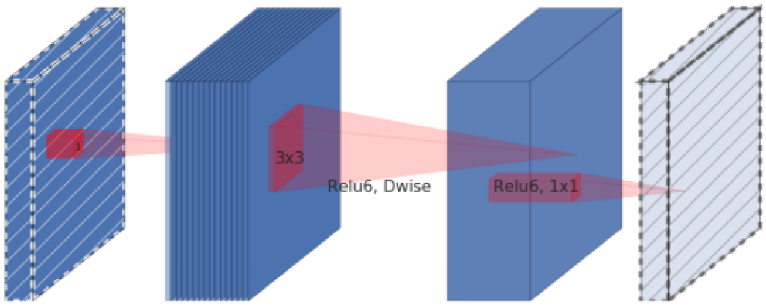
- Oddělení výpočtu pro různé kanály vstupního obrázku.
- Výpočet hlubokých (se zachováním všech kanálů určitého pixelu) map příznaků.
- Sumarizace příznaku pro všechny pixely.

Použití této vrstvy dokáže zmenšit počet výpočtů, které je potřeba provést pro inference[10] neuronové sítě. Zmenšení počtu výpočtů záleží na parametrech sítě, typicky je to osm až devětkrát méně. Přesnost přitom klesá pouze mírně [48].

Dalším významným rysem této architektury je použití ***Bottleneck konvolučních bloků s expanzní vrstvou***. Daná vrstva se skládá z:

- Expanzivní vrstvy – pointwise konvoluční vrstva[11] s velkým počtem kanálů. Na vstup daná vrstva dostává tensor dimenze $Df \times Df \times Cn$. Poté je výstupem tenzor $Df \times Df \times Cn \times t$, kde t je expanzní faktor - nový hyperperametr který ukazuje úroveň rozšíření vstupní dimenze.
- Depthwise konvoluční vrstva s aktivační funkcí $Relu6$.
- Klasická konvoluční vrstva s jádrem 1×1 a lineární aktivační funkcí (2.6.2).

Daná vrstva je zobrazena na obr. 2.25.



Obrázek 2.25: Architektura Bottleneck konvolučních bloků s expanzní vrstvou.

[10] Inferencemi neuronové sítí se zde rozumí proces, ve kterém neuronová síť ze vstupu vypočítá vlastní výstup.

[11] Pointwise konvoluce je typ konvoluce, který používá jádro 1x1: jádro, které iteruje každým bodem. Toto jádro má stejnou hloubku (počet kanálů), jako vstupní obraz.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Obrázek 2.26: Architektura MobilenetV2 kde:
Conv – konvoluční vrstva, *Conv dw* – hloubkově oddělitelná konvoluce, *Avg Pooling* – Sdružující vrstva která redukuje na základě funkce *average*(), *Dense* – plně propojená vrstva [48].

Kapitola 3

Vytváření aplikace

Cílem této práce je navrhnout a implementovat systém, který by umožňoval uživateli cvičit libovolnou pianovou kompozici, k níž má uživatel nahrávku. Daná kapitola popisuje klíčové požadavky na příslušný systém a způsob, jakým jsou tyto požadavky implementovány.

3.1 Návrh aplikace

3.1.1 Požadavky

Výsledná aplikace má dvě klíčové funkce:

1. Převod klavírní nahrávky do určité vnitřní reprezentace (viz kapitola 3.2.1).
2. Během cvičícího procesu kontrolovat v reálném čase to, co hraje uživatel.

První bod — transformace nahrávky do vnitřní reprezentace — můžeme implementovat za použití neuronové sítě z prací [23]. Přesnost jejích modelů je těžko dosažitelná v rámci diplomové práce a proto tento model (dále „velká neuronová síť“) bude téměř beze změn zahrnut do našeho systému. Nedostatkem velké neuronové sítě je malá rychlost (typicky zpracování vzorků délkou x sekund trvá $x \times 2$ sekund), což neumožní použití tohoto modelu ve druhém bodě. Proto za účelem implementace druhého bodu v dané práce bude použita jiná neuronová síť (dále „real-time neuronová síť“), jejíž cílem je rychlé poskytnutí klasifikace vstupních audio vzorků z mikrofonu.

3.1.2 Návrh řešení

Z hlediska uživatelé. Využívání aplikace uživatelem takto:

1. Uživatel otevře aplikaci a nahraje do ní nahrávku klavírní hudby.
2. Aplikace převede hudební nahrávku do vnitřní reprezentace.
3. Na základě vnitřní reprezentace aplikace začne ukazovat uživateli obdélníky, které reprezentují noty. Obdélníky jsou na začátku umístěny nahoře a postupně se budou snižovat. V okamžiku, kdy se jakýkoliv obdélník dotkne klávesnice, aplikace zastaví snižování obdélníků a bude čekat až uživatel zahraje na klavíru příslušnou klávesu. Poté bude aplikace opět pokračovat se snižováním obdélníků.

4. Pokud aplikace již nebude mít žádné obdélníky (na konci cvičícího procesu), aplikace ukáže uživateli pole s celkovým časem čekání aplikace na zmáčknutí jakékoliv klávesy. Dané pole bude ukázáno uživateli i v případě, pokud uživatel zmáčkne tlačítko *Stop*.

Příslušné GUI je znázorněno v příloze 7.1.

Z hlediska implementace. Cvičící proces z hlediska implementace obsahuje tři subprocesy: řídicí proces, proces poslouchání a rozpoznávání. Každý subproces má cyklus, který vykonává určitou hlavní činnost. Tyto činnosti jsou popsány níže.

- **Řídicí proces** - vykonává činnost popsanou v seznamu 3.1.2 v bodě 3. Klávesy, které byly zmáčknuty, tento proces najde v seznamu - $list_{prediction}$.
- **Proces poslouchaní** - tento proces pravidelně sbírá audio signál z mikrofonu, uplatňuje na tento signál filtr redukující šum (více 3.2.6), a ukládá výsledný signál do seznamu - $list_{listening}$.
- **Proces rozpoznávaní** - daný proces bude v cyklu:
  1. Načítat audio vzorky ze seznamu - $list_{listening}$.
  2. Převádět tyto audio vzorky na vstup do real-time neuronové sítě.
  3. Zapisovat výstup real-time neuronové sítě do seznamu $list_{prediction}$.

Všechny výše uvedené procesy začnou fungovat, až bude klavírní nahrávka převedena do vnitřní reprezentace.

3.2 Implementace aplikace

3.2.1 Vnitřní reprezentace not — note batches

V průběhu cvičícího procesu je potřeba rychle a bez zbytečných výpočtů generovat seznam not, které mají být vykresleny na obrazovce v následující okamžik. Z již existujících formátů, zaměřených na notový zápis, se MIDI jeví jako nejvhodnější. Při použití MIDI formatu je však potřeba projit MIDI soubor v několika cyklech:

1. Extrakce začátku a konce not.
2. Diskretizace času.

Proto byla navržena jiná metoda zápisu not obsažených v kompozici — note batches. Note batches představuje 2D seznam, kde je první dimenze tvořena časovou diskretizací: jsou to indexy, které představují časové kroky. Druhá dimenze představuje seznam not, které mají zaznít v tento časově diskretizovaný okamžik. Každá nota je reprezentovaná MIDI kódem, s tou výjimkou, že pokud je kód větší než 100, znamená to, že nota s kódem MIDI = (kód -100) má začátek v daný okamžik. Porovnání note batches, MIDI a notového zápisu můžeme vidět na Obr. 3.1.

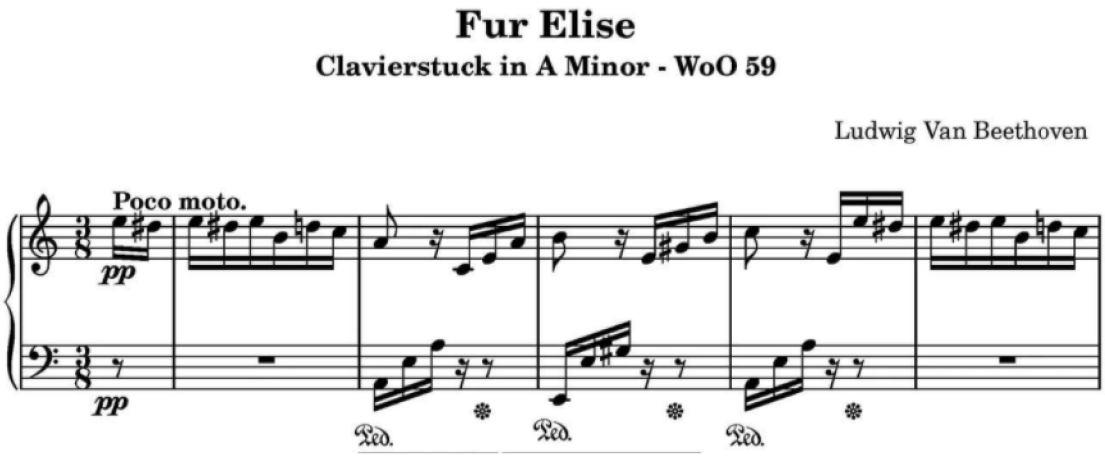
Tato notová reprezentace umožňuje rychle vykreslit poznámky na obrazovce. Vše, co musíme udělat, je načíst další řádek.

0	[132]
1	[32, 152]
2	[32, 52]
3	[32, 52]
4	[32, 52]
5	[]

Note batches

```
<message note_on channel=0 note=52 velocity=40 time=1521>
<message note_on channel=0 note=64 velocity=42 time=0>
<message note_on channel=0 note=59 velocity=43 time=211>
<message note_on channel=0 note=64 velocity=0 time=141>
<message note_on channel=0 note=64 velocity=44 time=0>
<message note_on channel=0 note=55 velocity=45 time=14>
<message note_on channel=0 note=59 velocity=0 time=126>
<message note_on channel=0 note=59 velocity=46 time=0>
<message note_on channel=0 note=52 velocity=0 time=141>
```

MIDI soubor



Hudebni zapis not

Obrázek 3.1: Ukázka různých způsobu reprezentace not.

3.2.2 Real-time neuronová síť

Cílem real-time neuronové sítě je poskytovat rychlou klasifikaci vstupních audio vzorků. K tomu, aby aplikace byla funkční, je potřeba kromě rychlosti dosáhnout navíc pro uživatele přijatelné přesnosti. V této kapitole je vysvětlen postup pro vytváření a trénování neuronové sítě, která by dokázala splnit tyto protichůdné požadavky.

Architektura neuronové sítě. Na výběru vhodné architektury závisí rychlost výsledné neuronové sítě i schopnost budoucí neuronové sítě zobecnit data, která uvidí v datasetu. Na výběru vhodné velikosti závisí schopnost neuronové sítě dosáhnout požadované přesnosti. Proto je zde důležité zvolit správnou architekturu a zvážit velikost. Avšak s rostoucí velikostí se model hůře trénuje. Kvůli tomu by bylo nejjednodušším řešením použít již trénovanou neuronovou síť na jiných datech a pomocí Transfer learningu (viz 2.5) přizpůsobit tuto síť naší aplikaci. Jako kandidáti byli zvoleni VGG16 a MobileNetV2. Důvody, proč jsou zvoleny právě tyto modely, jsou tyto:

- VGG16 je jednou z nejmenších konvolučních neuronových sítí, ale přesto je přesnost této sítě jen o málo horší než u větších analogických sítí.
- MobileNetV2 je konvoluční neuronová síť, která je primárně určena k běhu na mobilních zařízeních, a proto musí být velmi rychlá.

Po změně počtu neuronů výstupní vrstvy na 88 (aby sítě byly užitečné pro klasifikaci not), byly provedeny testy rychlosti. Na vstup byl 10000 krát podán stejný obrázek. Testy byly provedeny na následujícím počítači:

- Procesor: 2,7 GHz Intel Core i5
- RAM: 8 GB DDR3
- Operační systém: macOS Mojave

Výsledky jsou znázorněny v tabulce 3.1.

Model	Avg-t(s)	Max-t(s)	Min-t(s)
VGG16	0.34	1.8	0.2
MobileNetV2	0.05	0.29	0.027
Real-time NN	0.001	0.004	0.001

Tabulka 3.1: Porovnání rychlosti různých architektur neuronových síti

Nejbližší výsledky k real-time ukazuje MobileNetV2. Přestože minimální hodnoty stačí pro účely výsledné aplikace, průměrná hodnota bude způsobovat citelné zatížení systému. Zde je potřeba brát v potaz, že než neuronová síť začne zpracovávat vstup, na začátku je potřeba načíst z mikrofonu audio vzorky (32 ms) a provést redukci šumu a převést výsledný signál do Mel spektrogramu (více dále). Z tohoto důvodu rychlost výše uvedených neuronových sítí nepostačuje.

Vlastní neuronová sít. Z výše uvedených důvodů bylo rozhodnuto trénovat vlastní neuronovou síť od začátku. Jako možné kandidáty architektur můžeme považovat RNN a CNN. Pomocí frameworku Keras jsme vytvořili malé neuronové sítě uvedených typů. Na malém datasetu (podrobnosti v tabulce 3.3) jsme porovnali rychlost, s jakou dokáží trénovat. Bylo provedeno jedno menší trénování. Účelem daného trénování bylo zjistit, jak uvedené architektury neuronových sítí zvládnou data z naší datové sady. Jako limit délky trénování byol stanoveno 100 epoch. Výsledky můžeme vidět v tabulce 3.2

Model	Loss	Acc	Čas (m)
CNN	0.0034	0.5575	11
RNN	0.0041	0.5454	19

Tabulka 3.2: Porovnání trénování CNN a RNN, kde Loss - ztrátová funkce a Acc - přesnost modelu, Čas - čas v minutách, který byl věnován 100 epochám trénování

Typ datasetu	Počet trénovacích dvojic
Trénovací	1 722 685
Validační	215 335

Tabulka 3.3: Malý dataset.

Jak lze vidět, přesnost sítí je srovnatelná, ale délka trénování byla u RNN mnohem vyšší. Proto byla k dalšímu trénování zvolena pouze CNN síť. Výsledky testu rychlosti pro CNN jsou popsány v tabulce 3.1 v řádku *Real-time NN*.

Bylo vyzkoušeno několik architektur typu CNN. Jako nejvíce perspektivní se ukázala architektura založená na MobileNetV2. Kvůli nevýhodám BN byla daná vrstva vyloučena z budoucí architektury. Ke kompenzaci vyloučení BN byly uplatněny následující techniky, které měly přínos v několika různých článcích [50, 16]:

- Přidání více Residual spojení.
- Násobení větve (branch), která se vrací do hlavního grafu malou konstantou. V této práci byla použita konstanta 0.2.
- Použití počáteční inicializace vah - MSRA metodu [24].

Další změnou oproti MobileNetV2 bylo použití aktivační funkce LeakyRelu6 místo Relu6, která je definována následujícím vzorcem:

$$y = min(max(alpha \times x, x), 6), \tag{3.1}$$

kde *alpha* je malá konstanta (typicky mezi 0 a 1). V této práci byla *alpha* nastavena na hodnotu 0.3.

Výsledná architektura má dvě výstupní Dense vrstvy. První vrstva poskytuje 88 pravděpodobnost, že zazněl příslušný tón. Druhá vrstva poskytuje 88 pravděpodobnost, že síť selhala při zkoumání určitého tónu. Pomocí druhého výstupu dokážeme vyloučit z klasifikace odpovědi, ve kterých si neuronová síť není jista. Ztrátovou funkci druhé výstupní vrstvy byl $MSE(y', y_{error})$, kde y' je výstup první výstupní vrstvy a y_{error} je absolutní hodnota rozdílu y a y' (více kapitola 2.5). Zběžný přehled výsledné architektury je uveden v tabulce 3.4. Podrobná vizualizace je v příloze 7.3.1.

Typ vrstvy	Počet neuronů
Bottleneck	512
MaxPooling	2x2
Bottleneck	256
MaxPooling	2x2
Bottleneck	128
MaxPooling	2x2
Dropout	10%
Dense	256
Dense out 1	88
Dense out 2	88

Tabulka 3.4: Finální architektura real-time neuronové sítě. Bottleneck je vlastní varianta Bottleneck vrstvy z MobileNetV2. Vrstvy Dense out 1 a 2 jsou výstupní vrstvy a jsou napojeny na předchozí Dense vrstvu.

3.2.3 Vytváření datasetu

K tomu, abychom dokázali natrénovat vlastní neuronovou síť, je potřeba vytvořit dataset. Autoři [23] po trénování jejich neuronové sítě pokračovali ve sbíraní dat do nového datasetu – MAESTRO v2, který byl zveřejněn na podzim roku 2019. Tento dataset obsahuje 11 krát více dat pro trénování než předchozí dataset MARS (více kapitola porovnání datasetu 2.1) a má celkem 121.8 GB pianových nahrávek a příslušných MIDI souborů.

Příprava datasetu. Původně obsahuje dataset MAESTRO v2 *.wav* kompozice a příslušné *.MIDI* soubory. V této podobě je dataset téměř neužitečný. Proto jej musíme upravit tak, aby bylo možné na něm trénovat naši neuronovou síť. Úpravy se budou týkat:

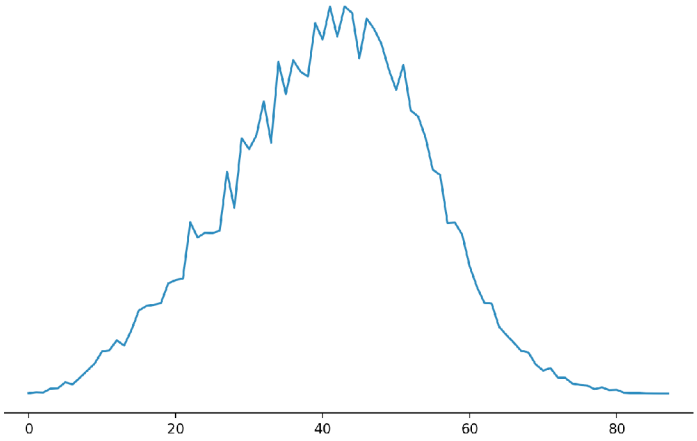
- Čištění dat.
- Transformace celých kompozic (.wav, .MIDI) do malých dvojic (.csv, .csv).

Čištění dat. Po analýze datasetu bylo odhaleno několik kompozic s jinými hudebními nástroji (housle). Tyto nástroje hrají zpravidla odlišné noty. V rámci trénování neuronové sítě pro klasifikaci pianových not by tyto odlišné nástroje pouze zvětšovaly šum a bránily by neuronové síti zobecnit užitečná data. Proto byl každý audiozáznam (.wav) v datasetu v poslechnout a kompozice s více nástroji byly eliminovány z datasetu. Celkem bylo eliminováno šest kompozic.

Také byla v datasetu odhalena velká disproporce not. Na Obr. 3.2 je zobrazen počet výskytu každé noty v datasetu.

Jak je vidět z Obr. 3.2, počet vzorků s notami od 25 do 60 je mnohem vyšší než ostatních. Tato skutečnost může způsobit úplné ignorování not s malým počtem výskytů. Kvůli faktu, že v rámci diplomové práce lze obtížně trénovat neuronovou síť s > 100 GB dat, bylo rozhodnuto zmenšit dataset. Avšak zmenšování bude probíhat takovým způsobem, který co nejvíce upraví distribuci počtu not v datasetu. V podstatě nebudeme tento dataset redukovat, ale spíše vytvoříme nový dataset ze starého.

Algoritmus pro vytváření nového datasetu:



Obrázek 3.2: Distribuce not v původním datasetu

1. Na začátku je dataset prázdný.
2. Všechny kompozice jsou tříděny podle poměru

$$\frac{N}{N'},$$

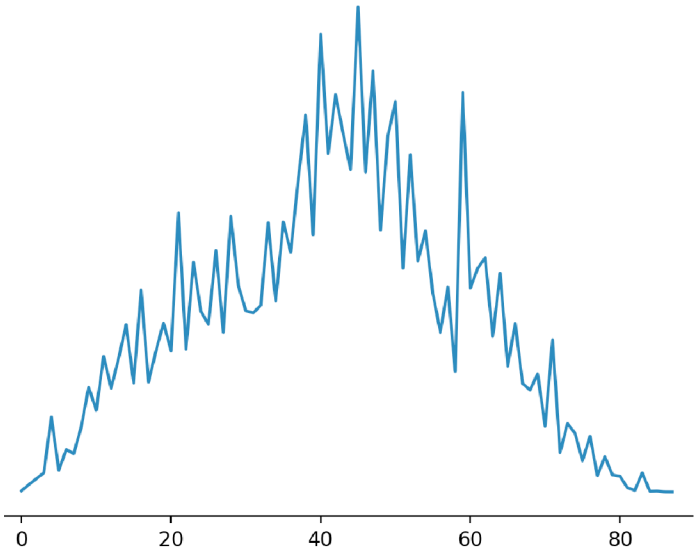
   kde N - celkový počet not, N' - počet not s MIDI kódem < 25 nebo > 60
3. Na každé dvojici souborů (.wav, .MIDI) pmusíme provést diskretizaci s krokem v 32 ms. Z tohoto dostaneme dvojici (PCM vzorky, seznam MIDI kódu).
4. Eliminovat zbytečné dvojice (více dále).
5. Přidat augmentované vzorky (více dále).
6. Pokud nebyl nepřevýšen limit v počtu vzorků, nebo v obsažené paměti pokračujeme, jinak přeskočíme na bod 8.
7. Pokračujeme od bodu 3.
8. Dataset je připraven.

Eliminace zbytečných dvojic. Dvojice (PCM vzorky, seznam MIDI kódu) se považuje za zbytečnou pokud:

- Seznam MIDI kódů je prázdný: dataset i bez těchto trénovacích dvojic obsahuje velkou množinu příkladů, kdy je nota potichu.
- V této dvojice teprve začala, nebo skončila hrát nota: 32 ms je velmi malé okno, ale může zachytit změnu, což bude znamenat, ze část PCM vzorku neodpovídá seznamu MIDI kódů.
- Následně musíme z našeho datasetu extrahovat co nejvíce různých vzorků. Avšak máme redukovat 120 GB dat do objemu, který bychom dokázali v rozumném čase zpracovávat. Tento limit velikosti byl stanoven na 14 GB. Proto byl do výsledného datasetu zařazen pouze každý pátý vzorek.

Augmentace dvojic. Jak je bylo uvedeno předtím, máme příliš velkou disproporci poctu not v datasetu. Proto kdy se nám povede najít dobry PCM vzorku které budou mít pouze noty které nám chybí, potom budeme chtít několikanásobně kopírovat tuto dvojici do našeho budoucího datasetu. Ale jednoduché kopírovaní taky může uškodit - neuronová sít bude podceňovat ostatní dvojice. Proto při kopírovaní trénovací dvojice která už je obsazena v datasetu, do PCM vzorku bude přidán malý aditivní bily šum.

Po provedeni výše uvedeného algoritmu nad původním datasetu dostali jsme nový dataset. Distribuce počtu not nového datasetu je na Obr. 3.3.



Obrázek 3.3: Distribuce not v novém datasetu

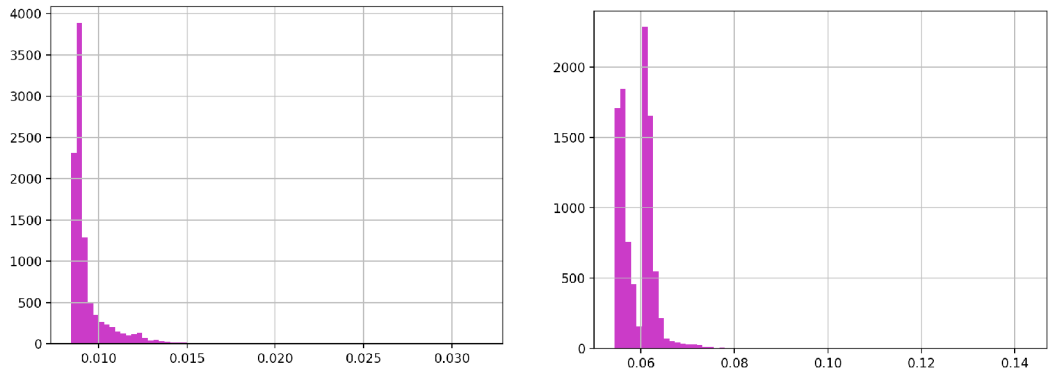
Jak lze vidět, tato transformace dokázala poněkud vyvážit distribuci počtu not.

Transformace dat. Pokud bude dataset obsahovat dvojici (audio vzorky, odpověď), neuronová síť se bude učit velmi dlouho. Typicky se za účelem zobecnit data aplikuje na audio vzorky některá z transformací: FFT, Mel-spec, CQT (více viz kapitola 2.1.4). V případě, že se jedná o klasifikaci zvuku, obvykle se používá Mel-spec, nebo CQT. K tomu, abychom zvolili správnou transformaci pro cílový dataset, byl proveden experiment rychlosti. 10000 náhodných vzorků bylo transformováno a normalizováno pro vstup do neuronové sítě. Výsledky jsou znázorněny na Obr. 3.4. Také byl proveden test přesnosti – byly vytvořeny dva malé datasety stejné velikosti jako 3.3. Jeden je s CQT transformací, druhý s Mel spektrogramem. Na každém datasetu 100 epoch byla trénována CNN. Výsledky jsou zobrazeny na 3.5

Jak lze vidět z Obr. 3.4 Mel spektrogram je mnohem rychlejší a více odpovídá požadavkům na real-time zpracování. I přesto, že daný druh transformace má horší výsledky na testu přesnosti 3.5, bude pro budoucí dataset využito právě Mel-spektrogram transformace.

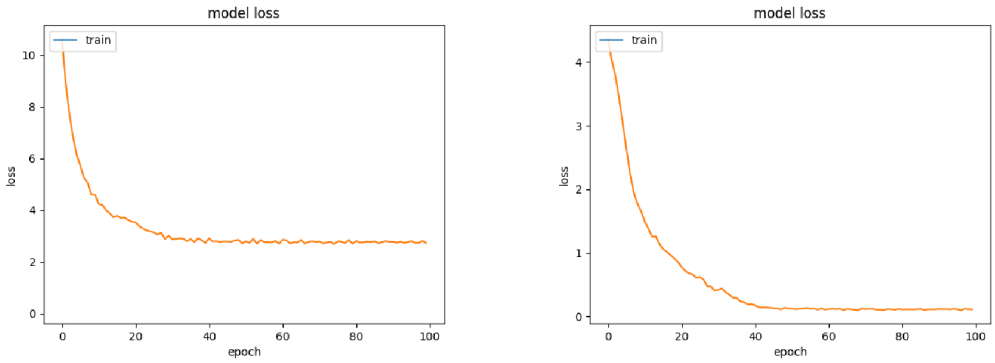
Jak bylo popsáno v teoretické části, algoritmy strojového učení lépe pracují na normalizovaných datech – rozsah hodnot vstupních dat musí být normalizován do malého rozsahu. Proto jsme vytvořili Mel-dataset a tento dataset byl normalizován – Mel koeficienty byly převedeny do jiného měřítka (od 0 do 1).

Následně byl dataset rozdělen do třech menších datasetů: trénovací, testovací a validační. Trénovací dataset byl používán pro trénování neuronové sítě. Na validačním datasetu byla



(a) (b)

Obrázek 3.4: Porovnání rychlosti transformace (b) – CQT, (a) – Mel-spektrogram.



(a) (b)

Obrázek 3.5: Porovnání ztrátové funkce během trénování na malých datasetech s různou transformací. (b) – CQT, (a) – Mel-spektrogram.

provedena průběžná kontrola, zda nedošlo k přeučení neuronové sítě (více dále). Kontrola byla prováděna jednou za pět epoch. Testovací dataset byl použit až na konci k hodnocení výsledné neuronové sítě. Počet trénovacích dvojic v každém datasetu je znázorněn v tabulce 3.5.

Typ datasetu	Počet trénovacích dvojic
Trénovací	9 034 030
Validační	639 381
Testovací	639 381

Tabulka 3.5: Velký dataset.

3.2.4 Trénování real-time neuronové sítě

Trénování probíhalo na výpočetním klastru – METACENTRUM[1]. Výsledná síť byla trénována během pěti dnů. Za tento čas bylo provedeno 147 epoch trénovacího cyklu. Byl použit batch size 1024, optimalizátor *Adam* s počátečním nastavením learning rate $1e-4$. Míra učení byla vynásobena 0,7 každých 10 epoch, přičemž minimální learning rate byla nastavena na $1e-6$. Jako ztratová funckce byla použita MSE.

Aby se předešlo přetrénování, bylo zavedeno dočasné ukončení trénování v případě, že se ztrátová funkce nebude zlepšovat během deseti epoch. K takovému ukončení nedošlo. Vzhledem k tomu, že nedošlo k dočasnému ukončení, můžeme říci, že síť má potenciál se zlepšit při pokračování trénování i v budoucnu. Grafy ztrátových funkcí lze najít v příloze 7.3.2.

3.2.5 Grafická část

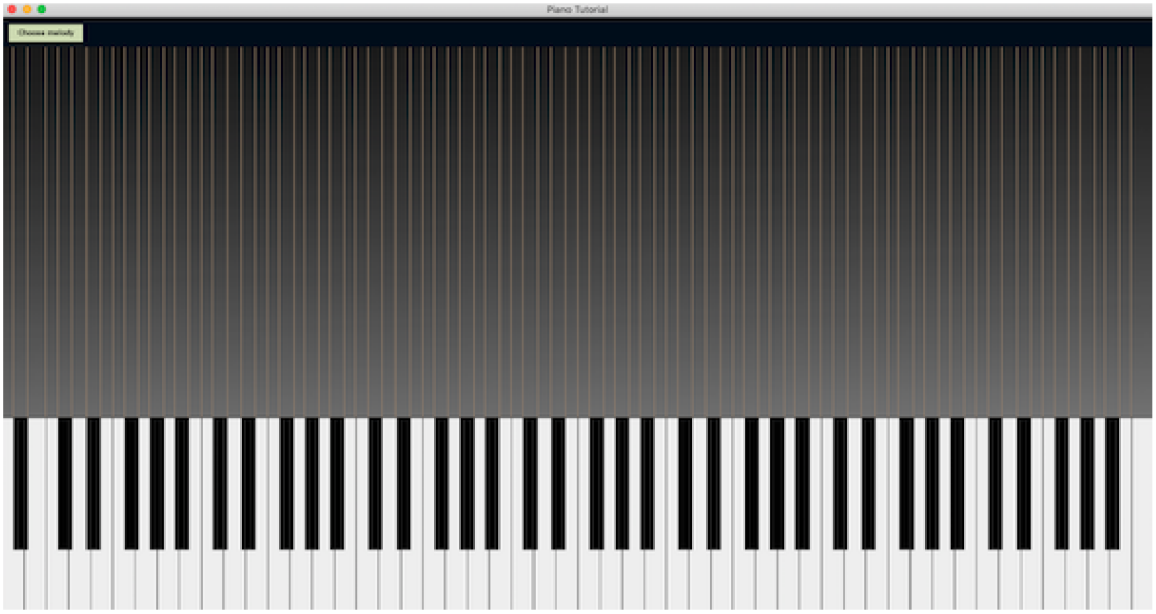
Grafická část je implementována s využitím knihovny Pygame. Z důvodu vzniklých problémů v již vyřešených modulech programu bude mít celková aplikace redukovanou funkcionalitu, a to za účelem robustního fungovaní přípustných modulů. Aplikace bude mít pouze jedno okno, které je znázorněno na Obr. 3.6.

Okno obsahuje pouze jedno tlačítko — *Choose melody*. Po zmáčknutí tlačítka se uživateli zobrazí obsah složky *songs* (budou zobrazeny pouze soubory: .midi, .mid, .wav a .csv). Pokud bude uživatel chtít cvičit kompozici z vlastního souboru, bude muset umístit vlastní soubor do této složky. Jestliže uživatel zvolí soubor, který se chce naučit, aplikace začne cvičící proces. Noty jsou reprezentovány barevnými obdélníky. Obdélníky postupně padají na příslušné klávesy. Když se obdélník dotkne klávesy, musí uživatel na vlastním klavíru stisknout příslušnou klávesu. Program pozastaví obdélníky, dokud uživatel nezmáčkne všechny klávesy, které má. Obdélník (klávesa), který má být stisknut, je zvýrazněná žloutou barvou. Když zmizí poslední obdélník (nota), proces cvičení se automaticky ukončí a uživateli se zobrazí celková doba čekání aplikaci na uživatele. Hodnota *celkov doba* představuje hodnocení uživatele. V budoucnu uživatel by se měl pokusit zlepšit dané hodnocení.

Uživatelský workflow můžeme zrekapitulovat do následujících bodů:

1. Uživatel zvolí nahrávku pianové kompozice.

[1] https://www.metacentrum.cz/cs/Sluzby/Grid/



Obrázek 3.6: Ukázka GUI

2. Aplikace na základě velké neuronové sítě převede tuto nahrávku do speciální vnitřní reprezentace – note batches.
3. Aplikace na základě reprezentace note batches začne cvičící proces.
4. Zmáčknutím tlačítka „mezera“ můžeme zastavit proces cvičení.
5. Opakovaným zmáčknutím tlačítka „mezera“ se proces cvičení prodlouží.

Stisknutím dalších tlačítek můžete ovládat proces cvičení. Seznam možných funkcí je v tabulce 3.2.5.

1. *<- 10 sec* - přeskočit o 10 sekund zpět.
2. *10 sec ->* - přeskočit vpřed o 10 sekund.
3. *speed down* - zpomalení.
4. *speed up* - zrychlení.
5. *normal speed* - obnovit normální rychlost.
6. *Stop* - ukončit trénování.
7. *Pause/Play* - zastavit/pokračovat trénování.

Všechny obrázky GUI v každé fázi aplikace jsou v příloze 7.2 a 7.3.

3.2.6 Předzpracování vstupního signálu, redukce šumu.

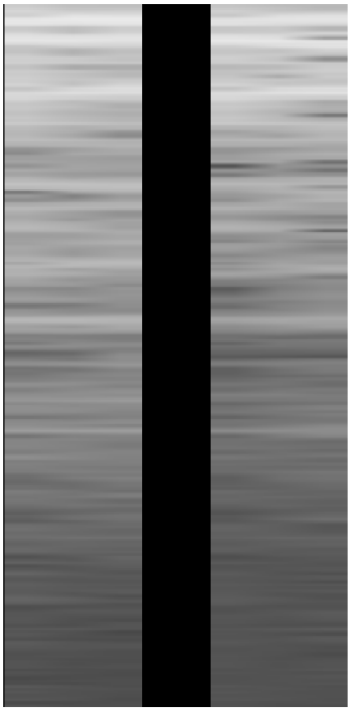
Tato aplikace je určena pro spouštění na vlastním počítači, nikoliv na profesionálním přístroji ve studiu. Proto existuje velké riziko nalézt ve vstupním signálu z mikrofonu mnoho šumu. Vliv tohoto šumu je potřeba redukovat. Za tímto účelem je v aplikaci použita redukce šumu na základě FFT (viz kapitola 2.1.4). Algoritmus filtrace šumu je následující:

1. Vypočítá se FFT u vzorového šumu.
2. Ve frekvenční doméně se vypočítají statistiky šumu: průměr a směrodatná odchylka.
3. Vypočítá se práh šumu pomocí vzorce:

$$y = mean(x) + \sigma(x) * k, \tag{3.2}$$

   kde x vstupní FFT je parametr udávající maskovací sílu masky.
4. Na základě prahu pro každou frekvenci se vypočítá maska a následně se lehce roztahuje.
5. Vypočítá se FFT u vstupního signálu.
6. Na spektrum vstupního signálu je aplikována maska.
7. Maskované spektrum se invertuje.

Vliv použití daného filtru na reálný audio signál lze vidět na Obr. 3.7.



Obrázek 3.7: Porovnání spektrogramu původní nahrávky z mikrofonu (vlevo) a nahrávky s redukovaným šumem (vpravo).

3.2.7 Volba užitečného frameworku pro strojové učení.

Pro trénování neuronových sítí existuje mnoho ML(strojové učení) framworku. Nejrozšířenější jsou Keras a Pytorch. Pytorch je framework pro strojové učení od Facebooku, který je zaměřen spíše na zkoumání: může dynamicky měnit architekturu během trénování, ale

inference[2] sítě prochází pomaleji než u Kerasu. Nevýhodou Pytorchu je i to, že nemá implementované multiprocesingové trénování. Implementovat samostatné multiprocesingové trénování neuronové sítě může výrazně zkomplikovat vytváření aplikace. Proto byl jako trénovací framework zvolen Keras. Keras má také výrazný nedostatek — pokud bude neuronová síť provádět inference modelu, Keras bude načítat celou neuronovou síť do operační paměti znovu od začátkua. Načítaní typicky zabere půl vteřiny. Toto bude negovat veškeré úsilí o výběr správné architektury. Řešením daného problému je převod Keras modelu do ONNX (Open Neural Network Exchange). ONNX je knihovna, která se používá pro výměnu modelů mezi různými ML frameworky. Kromě toho má však ONNX i vlastní inference. Tyto inference nepotřebují pokaždé načítat neuronovou síť do operační paměti (stačí pouze jednou, na začátku). Daný fakt výrazně zkrátí časovou zátěž chodu výsledné aplikace.

3.3 Zdrojové kódy.

Všechny zdrojové kódy jsou napsány s využitím jazyka Python 3.7. Odevzdané kódy mají následující strukturu:

- Applikation
- Training
- Dataset
- Test

Application. Daná složka obsahuje aplikační skripty. Hlavní skripty jsou:

- main.py – hlavní skript, který spustí aplikaci. Obsahuje všechny hlavní smyčky GUI a spouští všechny paralelní procesy.
- gui.py – obsahuje veškerý kód spojený s grafikou a vykreslováním objektů a obnovením displeje.
- Model.py – obsahuje třídy (interface) pro práci s neuronovými sítěmi.
- params.py – obsahuje všechny globální konstanty.
- prepare.sh – daný skript nainstaluje všechny potřebné knihovny pro spuštění aplikace.

Training. Daná složka obsahuje skripty spojené s trénováním malé neuronové sítě. Hlavní skripty jsou:

- train.py – skript pro spuštění trénování.
- data.py – skript pro přípravu a manipulaci s daty.
- models.py – skript pro definování různých architektur neuronových sítí.
- params.py – obsahuje všechny globální konstanty.

[2]Inference je část programu, která dokáže spustit neuronovou síť na základě souborů, kde je daná síť zapsána.

Dataset. Daná složka obsahuje skripty spojené s přípravou datasetu pro trénování z datasetu MAESTRO. Hlavní skripty jsou:

- preprocessingMAESTRO.py – skript pro spuštění přípravy datasetu.
- myMidi.py – skript, který kontroluje kvalitu vytvořeného datasetu.
- validateDataset.py – skript, jenž zkoumá vytvořený dataset.
- analizeDataset.py – skript který zkouma vytvořený dataset.
- params.py – obsahuje všechny globální konstanty.

Test. Daná složka obsahuje skripty pro testování trénované neuronové sítě. Hlavní skripty jsou:

- test.py – skript se všemi testovacími metodami.
- params.py – obsahuje všechny globální konstanty.

Kapitola 4

Experimentální vyhodnocení

Tato kapitola se zabývá experimentálním vyhodnocením vytvořené aplikace.

4.1 Časové testy

V tabulce 4.1 jsou znázorněny časy běhu jednotlivých modulů. Tabulka obsahuje i kumulované časy, které udávají přehled o celkové rychlosti aplikace. Jak lze vidět z tabulky

Modul	Avg-t(s)	Max-t(s)	Min-t(s)
Překreslování	0.007	0.085	0.003
Mel-transformace	0.045	0.109	0.009
Normalizace	0.0007	0.002	0.0005
Rozpoznavání	0.002	0.006	0.001
Zvuk (komplet)	0.0477	0.117	0.0105

Tabulka 4.1: Rychlost jednotlivých modulů aplikace, kde: *Překreslování* – kompletní čas vykreslování nové obrazovky (GUI loop), *Zvuk komplet* – čas pro zpracování zvuku (bez načítaní mikrofonu), *Mel-transformace* - čas potřebný pro transformaci zvukového signálu pomocí Mel-transformace, *Normalizace* – čas potřebný pro normalizaci výsledku Mel transformací, *Rozpoznavání* – čas potřebný pro inference neuronové sítě.

průměrná doba od začátku poslechu do klasifikace audio vzorků je *Zvuk (komplet) + 32ms = 0.0797* sekund, což můžeme považovat za téměř reálný čas. Taková rychlost příliš nezatěžuje systém kvůli tomu, že grafika se vykresluje nezávisle na zpracování zvuku. V případě, kdy modul zpracování zvuku nestihne předat rozpoznané noty do modulu GUI, GUI bude používat rozpoznané noty z minulého okamžiku. Tímto aplikace dokáže dosáhnout rychlosti v průměru 142 FPS[1], pokud bude rozpoznávání audio signálu pracovat s rychlostí 12.5 FPS.

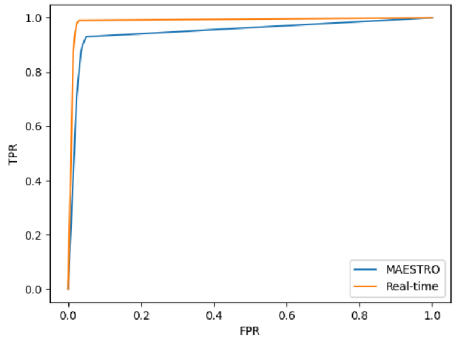
4.2 Přesnost klasifikaci

Testování na validačních datech. Po trénování byla malá neuronová síť testována na datech, které nebyly použity v trénovací ani v testovací části. Různé metriky přesnosti jsou znázorněny v tabulce 4.2. Z tabulky plyne, že síť dělá relativně málo chyb (*Loss=0.015*),

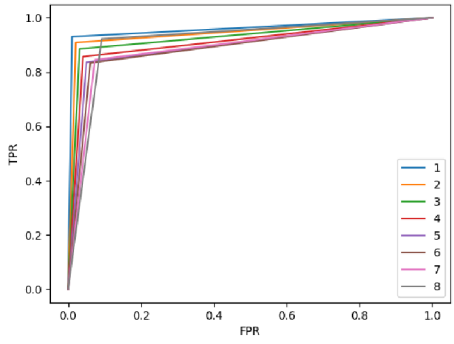
[1]FPS – Snímková frekvence (z anglického frames per second)

Metrika	Hodnota
Loss (MSE)	0.015
Accuracy	0.32
f1	0.81
precision	0.87
recall	0.76

Tabulka 4.2: Různé metriky přesnosti výsledné real-time neuronové sítě.



Porovnání malé (real-time) a velké (MA- ESTRO) neuronové sítě pomocí ROC křivek.



ROC křivky, které znázorňují porovnání kvality klasifikace vstupních vzorků podle počtu not ve vstupním vzorku.

Obrázek 4.1: Výsledky testování.

ale také vyplývá, že síť neprodukuje jisté odpovědi na vstupní data (*f1=0.81*). Pravděpodobně je to dáno příliš krátkou dobou trénování a použití Mel-transformace místo CQT při předzpracování vstupního signálu. Tento problém můžeme ve výsledné aplikaci vyřešit zmenšeným prahem jistoty, kdy se nota považuje za detekovanou.

Porovnání výsledků testů s předchozí prací. Daná síť byla otestována pomocí metrik ROC křivek. Dle těchto křivek můžeme dobře porovnat klasifikační modely. Křivka, která se více přibližuje k levému hornímu rohu, bude lepší než křivka, která je blíže diagonále mezi levým dolním rohem a pravým horním rohem. Tato diagonála reprezentuje náhodný klasifikátor. Na Obr. 4.1(vlevo) jsou zobrazeny ROC křivky pro malou a velkou neuronovou síť. Z obrázku plyne, že malá neuronová síť má menší přesnost, než velká neuronová síť. Avšak tento rozdíl není příliš významný. Na Obr. 4.1(vpravo) jsou zobrazeny ROC křivky pro různý počet not ve vstupním vzorku. Zde lze vidět, že se zvýšením počtu not ve vstupním audio vzorku přesnost klasifikátoru klesá. Což může způsobit problém při hraní složitějších kompozic.

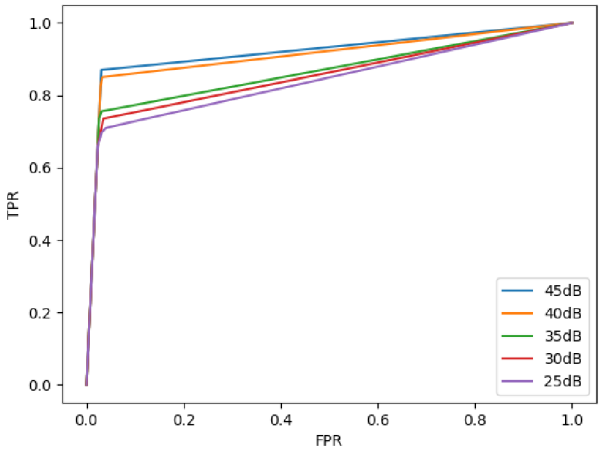
Omezení. Za účelem odhalení toho, jak dobře bude síť fungovat v reálném prostředí (místnost se šumem), byla malá neuronová síť testovaná s různým SNR poměrem. Kde SNR se vypočte dle následujícího vzorce [54]:

$$SNR(dB) = 10 \times \log_{10} \frac{P_{signal}}{P_{noise}}, \tag{4.1}$$

kde P_{signal} je průměrný výkon signálu a P_{noise} je průměrný výkon šumu. Dalo by se říci, že čím vyšší hodnota SNR, tím čistější bude signál. Testovací vzorky byly vygenerovány následovně:

1. Byly nalezeny všechny vzorky se třemi zaznělými tóny.
2. Na vlastním počítači byl nahrán šum chladičů.
3. Ke vstupním vzorkům z bodu 1 se aditivně přidával šum tak, aby SNR výsledného signálu odpovídal požadované hodnotě.

Výsledky testů jsou znázorněny na Obr. 4.2. Z časových důvodů bylo za účelem zkoumání vlivu šumu na stabilitu modelu použito méně různých prahů. Proto má graf menší počet různých bodů. I přesto je výsledný graf dostatečně informativní. Je zde vidět, že čím výše



Obrázek 4.2: Porovnání přesnosti sítě podle různého poměru SNR.

je úroveň šumu, tím horší výsledky jsou klasifikátoru.

4.3 Hodnocení GUI

Aplikace byla poskytnuta deseti lidem s různou úrovní dovednosti hry na klavír. Subjektivní hodnocení můžeme shrnou do následující pasáže: „Aplikace na učení hry na klavíru je celkem funkční. Začíná fungovat špatně při zvýšení úrovně šumu v prostředí kolem. Chybí možnost opakovat segmenty kompozice.“ Také je potřeba zmínit, že profesionální hráči měli potíže s rychlým hraním. Od určité rychlosti aplikace přestává stíhat sledovat hru uživatele a tím přestává být užitečná pro trénování hry na profesionální úrovni.

Ve srovnání s aplikacemi z 2.2.2 můžeme říci, že výsledná aplikace funguje přibližně na stejné úrovni přesnosti. Explecitním vylepšením je schopnost využít jakoukoli pianovou nahrávku.

Kapitola 5

Budoucí práce

Představená aplikace potřebuje několik vylepšení, například tyto:

- Výběr jiné frekvenční reprezentace zvukového signálu nebo optimalizace již existujících řešení pro výpočet CQT.
- Zvětšení přesnosti real-time neuronové sítě. Zejména pomocí zvětšení počtu skrytých konvolučních vrstev v real-time neuronové síti, delšího trénovacího procesu a zvětšení datasetu.
- Přidání možnost opakovaně cvičit pouze část kompozice.
- Přidání možnosti zápisu kompozice pro cvičení přímo v aplikaci.
- Přidání rozpoznávání více formátů (zatím aplikace umí pouze .wav a .MIDI).
- Přidání možnosti upravovat kompozici a ukládat ji do MIDI formátu.

Kapitola 6

Závěr

Cílem této diplomové práce bylo vytvoření aplikace pro podporu cvičení hry na klavír. Návrh takové aplikace představuje nelehkou úlohu, zejména kvůli následujícím dvěma problémům. Prvním problémem je přesný překlad zvukového záznamu uživatele do vnitřní reprezentace, na základě které bude uživatel cvičit. Druhým problémem je pořízení zvuku z mikrofonu a kontrola v reálném čase, jestli to, co uživatel hraje, odpovídá originální nahrávce, aby aplikace případně včas zastavila cvičení. Hlavním rozdílem mezi prvním a druhým problémem je nárok na přesnost a rychlost. V prvním případě je důležitá přesnost transkripce hudby do not, zatímco rychlost zpracování kritická není. Oproti tomu je ve druhém případě při kontrole uživatele v reálném čase doba zpracování jednoho zvukového fragmentu na prvním místě. Oba problémy byly redukovány na problém klasifikace přehrávaných not dle fragmentu zvukového záznamu.

Naše řešení bylo založeno na použití dvou různě velkých neuronových sítí (každá je použita pro jednotlivé části programu) s různými architekturami. První neuronová síť je zaměřena na přesnou transkripci zvukové nahrávky do vnitřní reprezentace notového zápisu – note batches. Tato síť byla s malými změny převzata z již existující práce [23]. Druhá síť byla vytvořena přímo autorem práce a byla použita za účelem průběžné kontroly zvuku z mikrofonu během cvičícího procesu. Velká část práci se zabývala vytvořením vlastní real-time neuronové sítě: výběru architektury, formátu vstupních dat, přípravou datasetu a trénováním.

Za účelem ověření přípustnosti nově navrženého řešení byla provedena série experimentů. Aplikace byla testovaná z časového hlediska i z hlediska použitelnosti koncovým uživatelem. Pomocí těchto testů byla ověřena vhodnost zvolených architektur neuronových sítí. Testy také ukázaly, že za určitých podmínek (omezení) je aplikace funkčním a užitečným pomocníkem při hře na klavír. Do budoucna lze aplikaci jistě vylepšit o další funkce a odstranit některé nedostatky, ovšem již v nynější podobě lze aplikaci využívat k danému účelu.

Literatura

[1] *Flowkey: Learn piano.* [online]. [cit. 2021-05-12]. Dostupné z: https://play.google.com/store/apps/details?id=com.flowkey.app&hl=en_IN.

[2] *Flowkey: Learn piano.* [online]. [cit. 2021-05-12]. Dostupné z: https://play.google.com/store/apps/details?id=com.joytunes.simplypiano&hl=en_IN.

[3] *Melody Scanner* [online]. [cit. 2021-05-12]. Dostupné z: https://melodyscanner.com/.

[4] *Music Transcription, anytime, anywhere* [online]. [cit. 2021-05-12]. Dostupné z: http://www.musictransapp.com/.

[5] *Synthesia* [online]. [cit. 2021-05-12]. Dostupné z: https://synthesia.app/.

[6] *Transcribe* [online]. [cit. 2021-05-12]. Dostupné z: https://www.seventhstring.com/xscribe/overview.html?__c=1.

[7] *Understanding ROC (Receiver Operating Characteristic) Curve | What is ROC?* [online]. [cit. 2021-05-02]. Dostupné z: https://www.mygreatlearning.com/blog/roc-curve/.

[8] *ISPRA'09: Proceedings of the 8th WSEAS International Conference on Signal Processing, Robotics and Automation.* Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2009. ISBN 9789604740543.

[9] Anonymous. *Activation Functions.* [online]. 2017 [cit. 2020-07-31]. Dostupné z: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html#elu.

[10] Anonymous. *Spectrogram of a piano composition.* [online]. 2017 [cit. 2020-07-31]. Dostupné z: https://www.reddit.com/r/dataisbeautiful/comments/5n16gp/spectrogram_of_a_piano_composition_you_can_see/.

[11] Anonymous. *Types of Neural Network Activation Functions.* [online]. 2017 [cit. 2020-07-31]. Dostupné z: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

[12] Belikova, K. *Unsupervised vs supervised learning .* [online]. 2018 [cit. 2020-07-31]. Dostupné z: https://neurohive.io/ru/osnovy-data-science/obuchenie-s-uchitelem-bez-uchitelja-s-podkrepleniem/.

[13] Benetos, E., Dixon, S., Duan, Z. a Ewert, S. Automatic Music Transcription: An Overview. *IEEE Signal Processing Magazine.* Leden 2019, sv. 36, s. 20–30.

[14] Benetos, E., Dixon, S., Duan, Z. a Ewert, S. Automatic Music Transcription: An Overview. *IEEE Signal Processing Magazine*. 2019, sv. 36, č. 1, s. 20–30. DOI: 10.1109/MSP.2018.2869928.

[15] Duhamel, P., Piron, B. a Etcheto, J. M. On computing the inverse DFT. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. 1988, sv. 36, č. 2, s. 285–286.

[16] Gaur, D., Folz, J. a Dengel, A. Training Deep Neural Networks Without Batch Normalization. *CoRR*. 2020, abs/2008.07970. Dostupné z: https://arxiv.org/abs/2008.07970.

[17] Golamina, I. P. *Sound*. Soviet encyclopedia, 1999. ISBN 5-85270-034-7. Dostupné z: http://www.femto.com.ua/articles/part_1/1222.html.

[18] Gold, O. a Sharir, M. Dynamic Time Warping and Geometric Edit Distance: Breaking the Quadratic Barrier. *ACM Trans. Algorithms*. New York, NY, USA: Association for Computing Machinery. srpen 2018, sv. 14, č. 4. DOI: 10.1145/3230734. ISSN 1549-6325. Dostupné z: https://doi.org/10.1145/3230734.

[19] Goodfellow, I., Bengio, Y. a Courville, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[20] Grass, J. *Data Science for beginners*. BXB, 2017. ISBN 978-5-9775-3758-2, 978-1-491-90142-7. Dostupné z: https://books.google.ru/books?id=9fzKivgcb30C.

[21] Hassan, M. ul. *VGG16 – Convolutional Network for Classification and Detection* [online]. 2018 [cit. 2020-07-31]. Dostupné z: https://neurohive.io/en/popular-networks/vgg16/.

[22] Hawthorne, C., Elsen, E., Song, J., Roberts, A., Simon, I. et al. *Onsets and Frames: Dual-Objective Piano Transcription*. 2017.

[23] Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A. et al. *Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset*. 2018.

[24] He, K., Zhang, X., Ren, S. a Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*. 2015, abs/1502.01852. Dostupné z: http://arxiv.org/abs/1502.01852.

[25] I, L. I. *Recognition of music compazitions.* [online]. 2016 [cit. 2020-07-31]. Dostupné z: https://elib.bsu.by/handle/123456789/160467.

[26] Jovanovic, J. *Shazam working principle.* [online]. 2016 [cit. 2020-07-31]. Dostupné z: https://habr.com/ru/company/wunderfund/blog/275043/.

[27] Kaarov, A. *WAV format.* [online]. 2019 [cit. 2020-07-31]. Dostupné z: https://audiocoding.ru/articles/2008-05-22-wav-file-structure/.

[28] Knuth, D. E. *The TEXbook*. Addison-Wesley Publishing Company, 1996. ISBN 0-201-13447-0.

[29] KRESTJANINOFF. *Speech recognition* [online]. 2014 [cit. 2021-04-23]. Dostupné z: https://habr.com/ru/post/226143/.

[30] LEE, D. a SEUNG, H. Algorithms for Non-negative Matrix Factorization. *Adv. Neural Inform. Process. Syst.* Únor 2001, sv. 13.

[31] LI SU, Y.-H. Y. Combining Spectral and Temporal Representations for Multipitch Estimation of Polyphonic Music. *IEEE Transactions on Electrical Insulation*. 2015, sv. 23, č. 10, s. 1600–1612.

[32] MCCULLAGH, P. a NELDER, J. *What is a softmax activation function?* [online]. 1989 [cit. 2020-07-31]. Dostupné z: http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-12.html.

[33] OKSUMORON. *Tuning loss functions.* [online]. 2020 [cit. 2020-07-31]. Dostupné z: https://habr.com/ru/company/ods/blog/488852/.

[34] PARMAR, R. *Common Loss functions in machine learning.* [online]. 2018 [cit. 2020-07-31]. Dostupné z: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[35] PLUTAX, P. *Apologize - One Republic.* [online]. 2016 [cit. 2020-07-31]. Dostupné z: https://www.youtube.com/watch?v=egIRUeXG0YA.

[36] PORTILL, R. *Understanding Dynamic Time Warping.* [online]. 2019 [cit. 2020-07-31]. Dostupné z: https://databricks.com/blog/2019/04/30/understanding-dynamic-time-warping.html.

[37] ROBERTS, A. *Onsets and Frames JS* [online]. 2018 [cit. 2021-05-12]. Dostupné z: https://piano-scribe.glitch.me/.

[38] ROEDERER, J. *The Physics and Psychophysics of Music: An Introduction*. Springer New York, 2008. ISBN 9780387094748. Dostupné z: https://books.google.cz/books?id=rYfqoc1dDmYC.

[39] SATISH, L. a GURURAJ, B. I. Use of hidden Markov models for partial discharge pattern classification. *IEEE Transactions on Electrical Insulation*. 1993, sv. 28, č. 2, s. 172–182.

[40] SCHINDLER, A., LIDY, T. a BÖCK, S. *Deep Learning for MIR Tutorial*. 2020.

[41] SCHUSTER, M. a PALIWAL, K. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*. Prosinec 1997, sv. 45, s. 2673 – 2681. DOI: 10.1109/78.650093.

[42] SCIENCE, O. D. *Metrics in Machine Learning.* [online]. 2017 [cit. 2020-07-31]. Dostupné z: https://habr.com/ru/company/ods/blog/328372/.

[43] SERGEY IOFFE, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv*. Únor 2015.

[44] STUART RUSSELL, P. N. *Artificial Intelligence: A Modern Approach*. Prentice Hall PressOne Lake Street Upper Saddle River, NJUnited States, 2013. https://dl.acm.org/doi/book/10.5555/1671238.

[45] Swift, A. *A brief Introduction to MIDI.* [online]. 1997 [cit. 2020-07-31]. Dostupné z: https://web.archive.org/web/20120830211425/http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/aps2/.

[46] Tan, A. *Neural Network (DIY).* [online]. 2020 [cit. 2020-07-31]. Dostupné z: https://www.mathworks.com/matlabcentral/fileexchange/67750-neural-network-diy.

[47] Therrien, C. a Tummala, M. *Probability and Random Processes for Electrical and Computer Engineers, Second Edition.* Taylor & Francis, 2011. ISBN 9781439826980. Dostupné z: https://books.google.ru/books?id=9fzKivgcb30C.

[48] Tsang, S.-H. *Review: MobileNetV2 — Light Weight Model (Image Classification)* [online]. 2019 [cit. 2020-07-31]. Dostupné z: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c.

[49] Valueva, M., Nagornov, N., Lyakhov, P., Valuev, G. a Chervyakov, N. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and Computers in Simulation.* 2020, sv. 177, s. 232 – 243. DOI: https://doi.org/10.1016/j.matcom.2020.04.031. ISSN 0378-4754. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0378475420301580.

[50] Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y. et al. ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks. *CoRR.* 2018, abs/1809.00219. Dostupné z: http://arxiv.org/abs/1809.00219.

[51] Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y. et al. ESRGAN: Enhanced super-resolution generative adversarial networks. In: *The European Conference on Computer Vision Workshops (ECCVW).* September 2018.

[52] Wittek, P. 5 - Unsupervised Learning. In: Wittek, P., ed. *Quantum Machine Learning.* Boston: Academic Press, 2014, s. 57 – 62. DOI: https://doi.org/10.1016/B978-0-12-800953-6.00005-0. ISBN 978-0-12-800953-6. Dostupné z: http://www.sciencedirect.com/science/article/pii/B9780128009536000050.

[53] Zhou, H., Melloni, L., Poeppel, D. a Ding, N. Interpretations of Frequency Domain Analyses of Neural Entrainment: Periodicity, Fundamental Frequency, and Harmonics. *Frontiers in Human Neuroscience.* Červen 2016, sv. 10. DOI: 10.3389/fnhum.2016.00274.

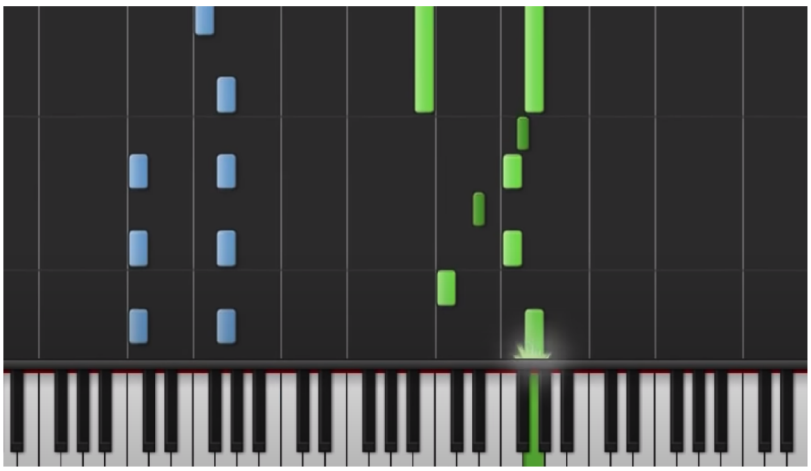
[54] Černocký, H. *Zpracování řešových signalů - studijní opora.* Vysoké učení technické v Brně, 2006. Dostupné z: https://docplayer.cz/12788883-Http-www-fit-vutbr-cz-cernocky.html.

Kapitola 7

Příloha

7.1 Grafické rozhraní

Inspirace Obr. 7.1. GUI výsledné aplikace na Obr. 7.2 a 7.3.



Obrázek 7.1: Příklad toho, jak aplikace Syntesia vypadá.

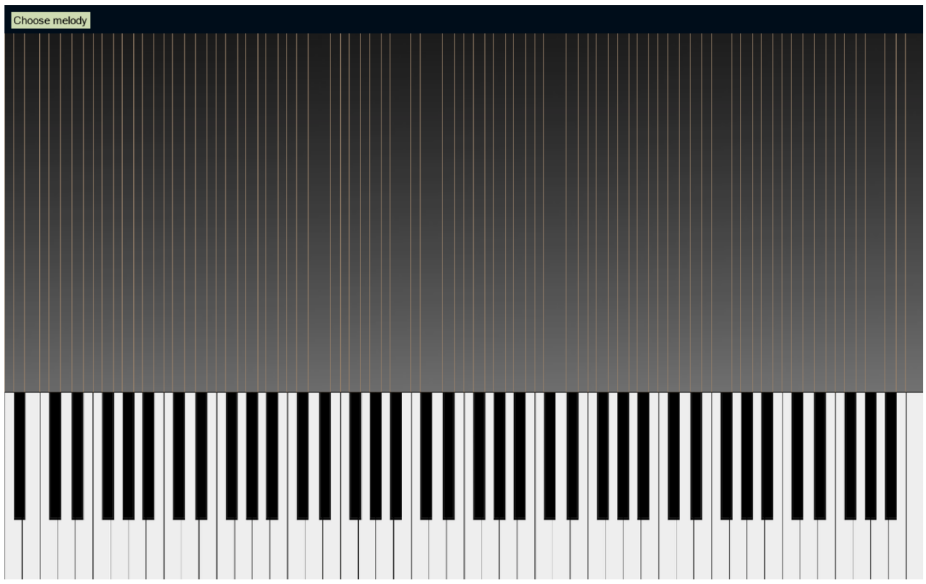
7.2 Batch normalization

Ukazuje se že čím víc vrstev obsahuje síť s BN, tím větší pravděpodobnost dostat na výstupu artefakty (Obr. 7.4 (a), (b)). Riziko vzniku artefaktů zvyšuje menší *batch size*. Vzniká kvůli nestabilnímu trénování – patche mají navzájem různé datové rozložení [51]. Na dolním obrázku (c) 7.4 lze vidět výskyt artefaktů v průběhu trénovací procesu.

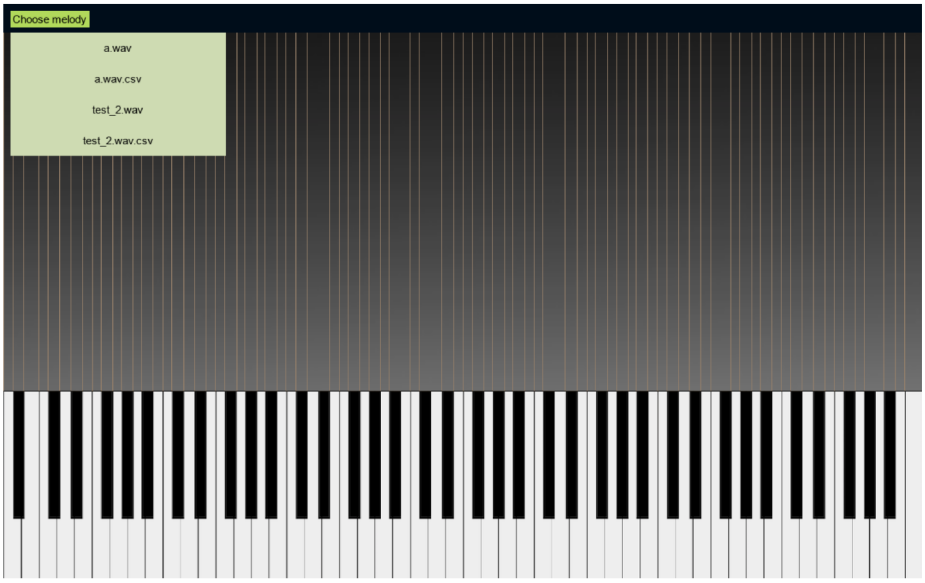
7.3 Real time neuronová síť

7.3.1 Architektura

Na obrázcích 7.5 a 7.6 je vizualizace použité architektury neuronové sítě.



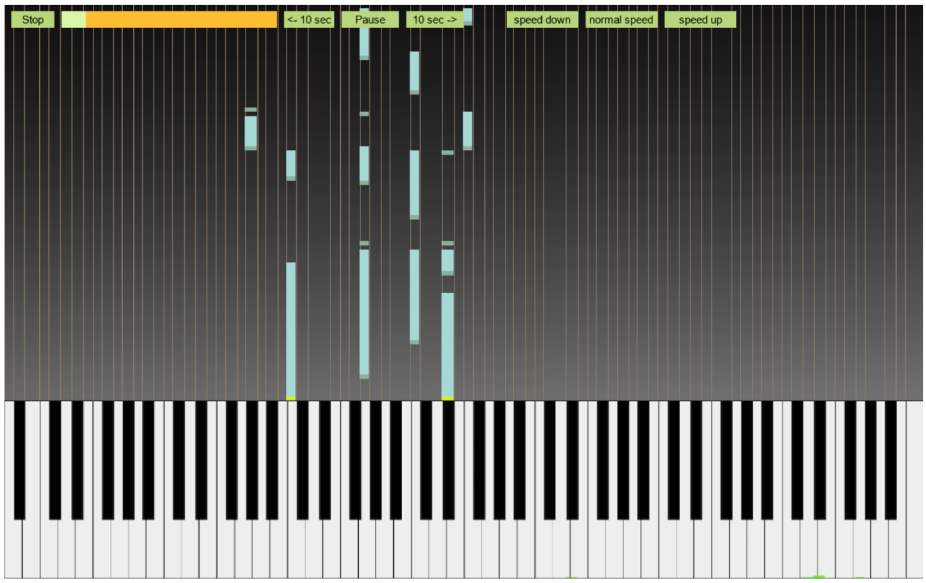
Hlavní okno.



Výběr souboru ke zpracování

Obrázek 7.2: Demonstrace každého kroku ve výsledné aplikaci (část 1).

.

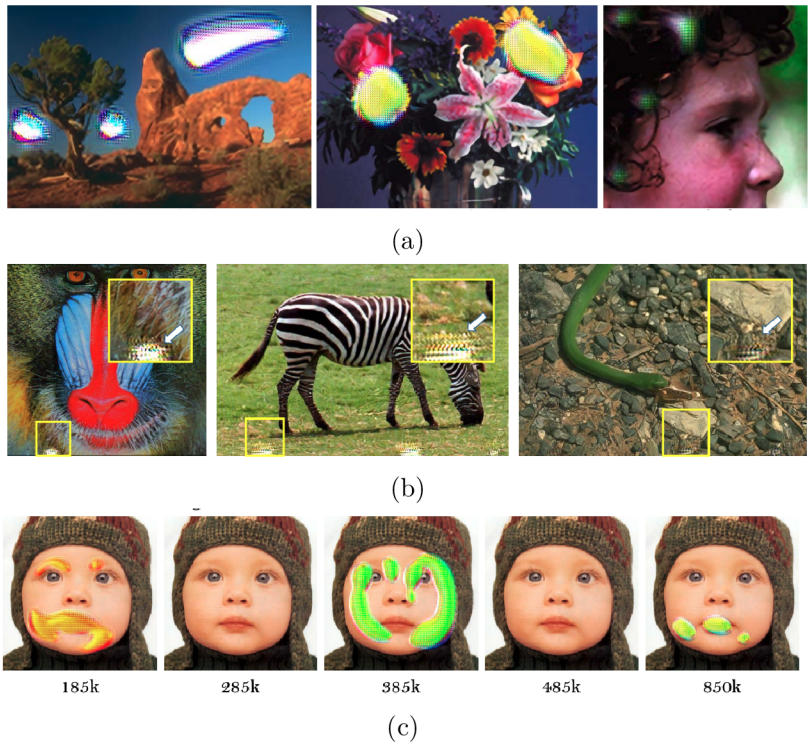


Proces cvičení.

Konec cvičení.

Obrázek 7.3: Demonstrace každého kroku ve výsledné aplikaci (část 2).

.



(a)

(b)

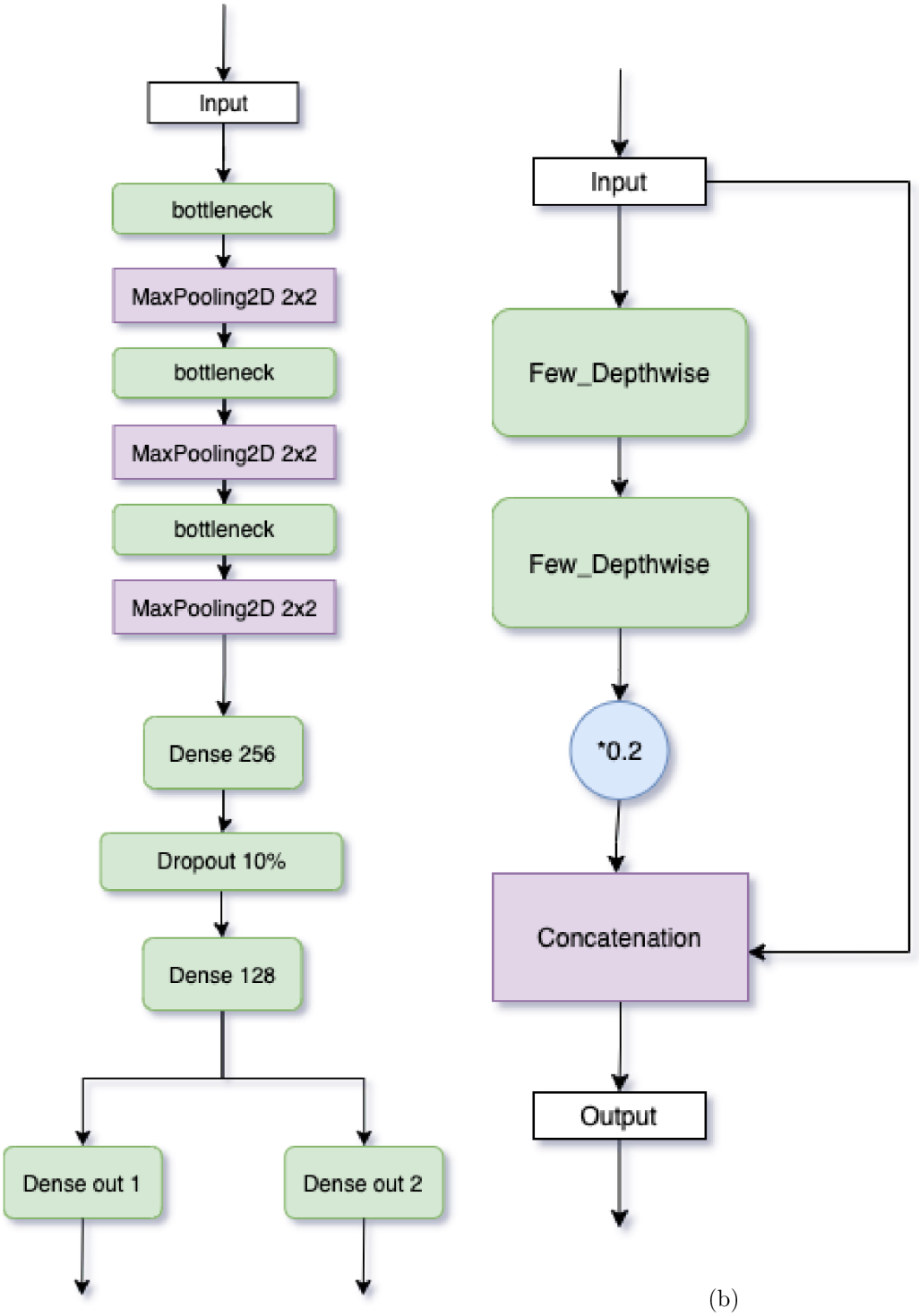
(c)

Obrázek 7.4: Ukázka artefaktů způsobených přidáním Batch Normalization vrstvy do neuronové sítě [51].

.

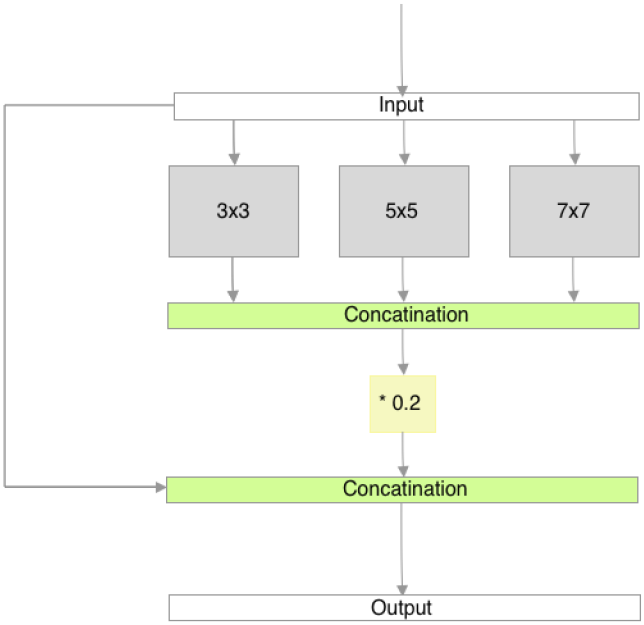
7.3.2 Trénování

Tato část představuje čísla týkající se průběhu trénování malé neuronové sítě. Obr. 7.7 zobrazuje grafy průběhu funkce ztráty za celý průběh trénování. Obr. 7.8 ukazuje grafy ztrátové funkce pro zkrácené trénování (ořízнuto prvních 50 epoch). Na těchto grafech je lepší vidět, zda se neuronová síť dokaže kanvergovat dál nebo ne.

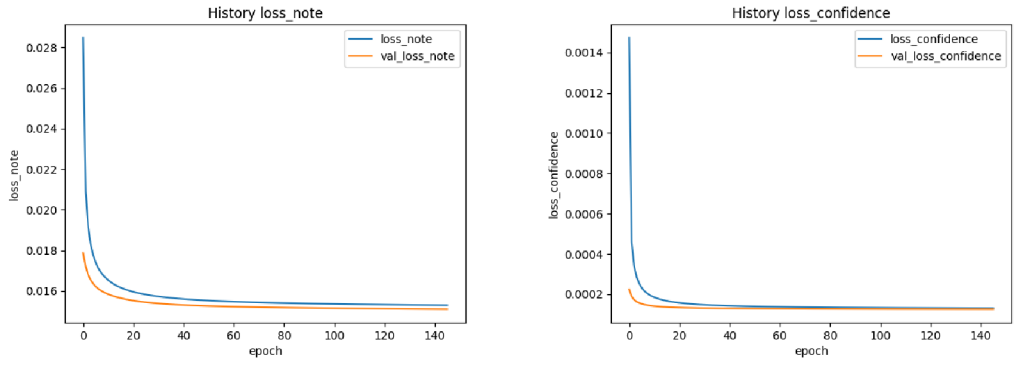


(a) (b)

Obrázek 7.5: Vizualizace konečné architektury malé neuronové síti. (a) - celkový přehled architektury, kde *Dense out 1* produkuje 88 pravděpodobnosti, že určitý ton zazněl ve vstupním vzorku a *Dense out 2* produkuje 88 pravděpodobnosti, že příslušná pravděpodobnost z *Dense out 1* je spolehlivá. (b) - vrstva *bottleneck*, kde *Few Depthwise* je vrstva která se skládá z několika Conv2D vrstev (znázorněná na Obr. 7.6) a **0.2* znamená vynásobení výstupu předchozí vrstvy hodnotou *0.2*.



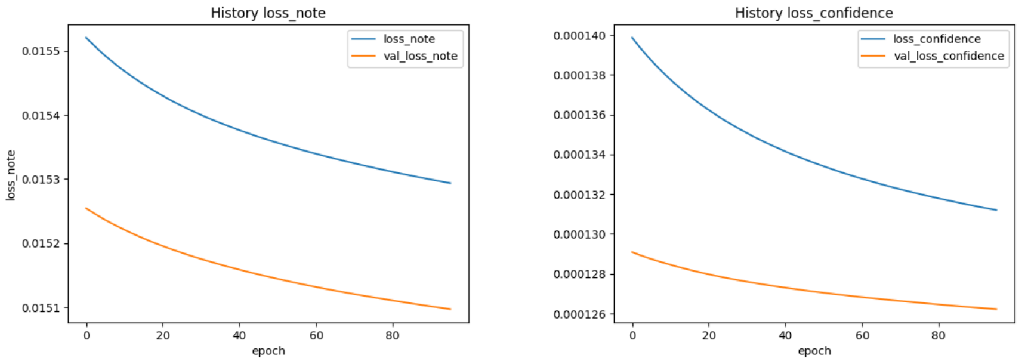
Obrázek 7.6: Vizualizce vrstvy *Few Depthwise*, kde šedé bloky znázorňují konvoluční vrstvy.



(a) (b)

Obrázek 7.7: Vizualizace ztrátových funkci za celé období trénování. (a) - ztrátová funkce pro výstup Dense-1, (b) - ztrátová funkce pro výstup Dense-2.

.



(a) (b)

Obrázek 7.8: Vizualizace ztrátových funkci za zkrácené období trénování. (a) - ztrátová funkce pro výstup Dense-1, (b) - - ztrátová funkce pro výstup Dense-2.

.