



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

AUTHOR

Bc. NIKITA USTINOV

VEDOUcí PRÁCE

SUPERVISOR

Dr. Ing. FRANTIŠEK ZBOŘIL, Ph.D.

BRNO 2020

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 v notovém zápise umožnil tento zápis digitálně zpracovat a následně 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

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

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 2019

Datum odevzdání: 31. července 2020

Datum schválení: 31. října 2019

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 zapsané v paměti. Aplikace která je výsledkem této práce vyřeší daný problém - uživatel bude moci nahrát do aplikace libovolnou klavírní nahrávku kterou chce cvičit a aplikace se postará o vytvoření procesu cvičení. Proces cvičení se spočívá v ukázce uživatelů určitým způsobem upravených not, a současná kontrola, pomocí mikrofону, co hraje uživatel. Největším výzvou dané diplomové práce bylo najít způsob jak přesně a rychle klasifikovat audio signál z mikrofónů. Daná úloha byla vyřešena pomocí dvou nezávislých neuronových sítí s různou architekturou, které byli trénovány na různých datových sadách. Za účelem odůvodnění zvoleného řešení budou uvedeny všechny potřebné teoretické muzikální a vědecké pojmy a metody, které mají k tomu přímý vztah. Výsledná aplikace bude testována z třech hledisek: přesnosti, rychlosti, uživatelské použitelnosti.

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 three respects: accuracy, speed, 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, 2020. 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 bakalářskou práci vypracoval samostatně pod vedením pana Dr. Ing. Zbořila, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....

Nikita Ustinov
31. července 2020

Poděkování

Děkuji vedoucímu práce Dr. Ing. Františku 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

1	Úvod	3
2	Teoretický základ	5
2.1	Zvuk a reprezentace zvuku	5
2.1.1	Ton, nota	5
2.1.2	Formáty audio	5
2.1.3	Frekvenční reprezentace zvuku	6
2.2	Modely pro transkripci zvuku	8
2.2.1	Neuron	9
2.2.2	Učení neuronových sítí	12
2.2.3	Ztrátová funkce	13
2.2.4	Vrstvy neuronových sítí	14
2.2.5	Architektury neuronových sítí	16
2.2.6	Příprava trénovací sady	20
2.2.7	Metriky trénování	21
2.3	Existující přístupy pro transkripci zvuku	22
3	Navrh systému	25
3.1	Notová reprezentace — note batches	25
3.2	Cvičící proces	27
3.3	Real-time neural network	27
3.3.1	Architektura neuronové sítě	27
3.3.2	Vlastní neuronová síť	28
3.3.3	Vytváření datasetu	28
3.4	Implementace	31
3.4.1	Volba užitečného ML frameworku	31
3.4.2	Předzpracování vstupního signálu, redukce sumu.	32
3.4.3	Trénování real-time neuronové sítě	34
3.4.4	Grafická část	34
4	Experimentální vyhodnocení	36
4.1	Časové testy	36
4.2	Přesnost klasifikaci	37
4.3	Hodnocení GUI	37
4.4	Omezení	37
5	Budoucí práce	38

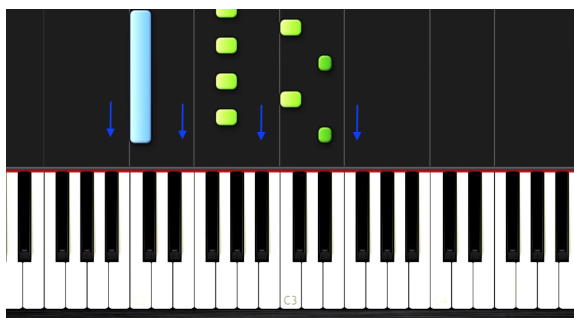
6 Závěr	39
Literatura	40

Kapitola 1

Úvod

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

- Uživatel nahraje do aplikace audio záznam klavírní hudby.
- Aplikace převede daný záznam do MIDI (nebo jiného vnitřního) formátu.
- Aplikace bude postupně dělat dvě věci: ukazovat uživateli noty které jsou reprezentovány barevnými obdélníky a na základě poslouchání mikrofonom kontrolovat co uživatel hraje na klavíru. Obdélníky postupně padají na klávesy v GUI. Když se obdélník dotkne klávesy, program zastaví padající klávesy a bude čekat pokud uživatel nezmáčkne příslušnou klávesu na klavíru. Po zmáčknutí padání bude pokračovat. Obr. 1.1 demonstruje možné GUI.



Obrázek 1.1: Možný UI **TODO** [?]

Takový cvičící proces může být velmi užitečný pro uživatele, který nezná muzikální teorie, noty a notový zápisu. Grafická reprezentace not ve tvaru padajících obdélníků může uživatele naučit kdy a jakou klávesy musí zmáčkout, aby zazněla správná melodie.

Zjednodušením od standardní úlohy automatizované transkripci hudby (automatic music transcription, AMT [1]) je skutečnost, že v dané aplikaci 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 klavír. Návrh takové aplikace představuje netriviální úlohu, zejména kvůli následujícím dvěma problémům. První vyzvou je přesný překlad zvukového záznamu uživatele do vnitřní reprezentace, na základe které pak bude uživatel cvičen. Druhým problémem je pořízení zvuku z mikrofону a kontrola v reálném čase zda to, co uživatel hraje, odpovídá originální nahrávce, aby aplikace včas zastavila padání not. 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 ve druhém případě, při real-time kontrole uživatele, je doba zpracování jednoho zvukového fragmentu na prvním místě. Oba dva problémy lze redukovat na problém klasifikace přehrávaných not podle fragmentu zvukového záznamu. Nejběžnějším způsobem klasifikace zvuku v těchto dnech jsou neuronové sítě **TODO cite**, skryté Markovské retezce **TODO cite** nebo techniky zpracování signálu (např. autokorelace **TODO cite**).

Naše řešení je založené na použití dvou různých velkých neuronových sítí (každá pro jednotlivé části programu) s různými architekturami. První neuronová síť bude zaměřena na přesnou transkripci nahrávky do notového zápisu. Taková síť se všemi kladenými na ní požadavky již existuje a byla zveřejněna v práci (6). Danou síť jednoduše integrujeme do výsledné aplikace. Druhá síť bude vytvořená vlastními silami a použita za účelem průběžné kontroly zvuku z mikrofону, během cvičícího procesu. Velká část dané práce se bude zabývat právě vytvořením vlastní real-time neuronové sítě: výběru architektury, formátu vstupních dat, přípravou datasetu a samotnému trénování.

Celkové řešení bude testováno zda odpovídá časovým požadavkům na real-time a zda je použitelné pro konečného uživatele.

Kapitola 2

Teoretický základ

Na začátek je potřeba formalizovat předmět našeho zájmu. Tato kapitola vymezuje všechny pojmy a principy, znalost kterých je nezbytná pro pochopení dané práce.

2.1 Zvuk a reprezentace zvuku

Zvuk [?] je fyzický fenomén, který se šíří ve formě elastických vln mechanických vibrací v prostoru. Stejně jako libovolná vlna zvuk se dá 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 dal)[2, 3].

2.1.1 Ton, nota

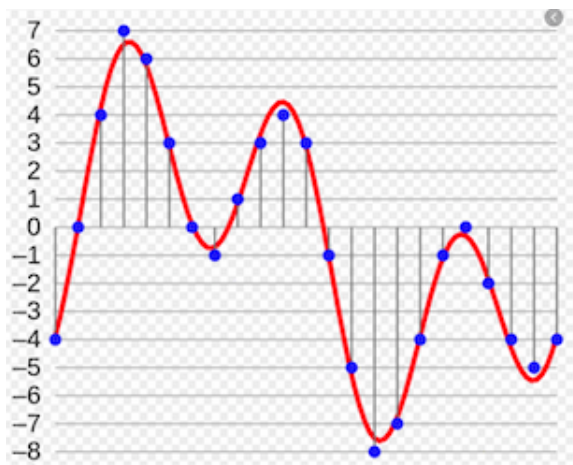
Ton je stálý zvuk s určitou frekvencí¹. Z kombinace tónů se skládá hudba. Notou nazýváme grafickou reprezentaci tonu, viz. Obr 3.1 (Hudební zápis not). Pro pojmenování not používáme následující symboly: C, D, E, F, G, A, B. Když frekvenční poměr dvou zvuku je stupněm dvojky, potom tyto zvuky vnímáme jako podobné a značíme stejným symbolem. Interval mezi těmito zvuky se nazývá oktáva. Každá oktáva se dále dělí na 12 intervalů které se nazývá půltóny. Pro rozlišení mezi stejnými noty různých oktáv se používají číslice, např. C1, D4 atd.[1,2] Síla tonu udává jeho hlasitost.

2.1.2 Formáty audio

WAV. Chceme-li zvuk zapsat nebo přehrát na nějakém elektronickém zařízení, je potřeba použít vhodný zvukový formát, který moderní zařízení dokážou pochopit. Základním formátem kterému rozumí všechny počítače s operačním systémem Windows je WAV formát. Ve své podstatě WAV představuje kontejner (obálka) nad určitým audio formátem, typicky nad formátem PCM (Pulse Code Modulation). PCM je docela starý formát, ale v něm se dá zapsat velmi kvalitny zvuk. PCM soubor obsahuje sekvenci vzorků v čase, kde každý vzorek udává amplitudu (sílu) signálu v určitý okamžik (viz Obr. 2.1). Nevýhodou daného formátu je jeho příliš velký rozměr. Právě tento problém se snažili vyřešit pozdější formáty jako například MP3.

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

¹V literatuře se často říká že tón je stálý zvuk s určitou výškou. Výšku ale se dá odvodit jednoduchými výpočty z frekvenci. Proto v textu jsme odvozujeme výšku od frekvenci.



Obrázek 2.1: PCM ukázka

hudebními nástroji. Na rozdíl od WAV, MIDI se 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. Na Obr. 2.2 je vidět standardní MIDI soubor.

```
<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>
```

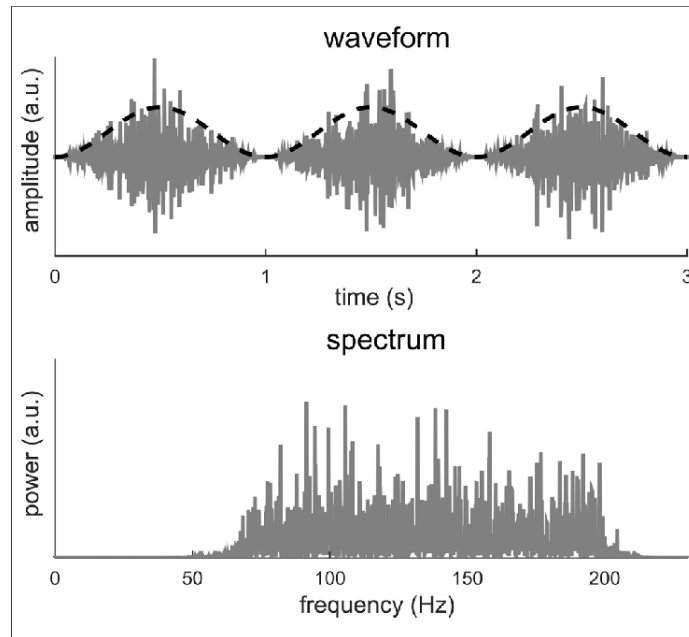
Obrázek 2.2: MIDI soubor je strukturován jako seznam událostí. Každá událost obsahuje značku , kód klávesy (notě), čas (time) a rychlost (velocity). Kód klávesy je to speciální MIDI kód který jednoznačně odkazuje na klávesu na hudebním nástroji. Čas události říká kolik časových jednotek uplynulo od předchozí události. Rychlost udává s jakou silou byla zmáčknutá klávesa. Pokud rychlost se rovná nule, tato událost odkazuje na skutečnost že klávesa byla uvolněna.

2.1.3 Frekvenční reprezentace zvuku

Těžko se dá 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. Za tím účelem můžeme použít Diskrétní Fourierovou Transformací (DFT). Necht $\{x_n\}_{0 \leq n \leq N}$ je vzorkovaný audio signál. k -ta složka jeho frekvenčního spektra, X_k , se počítá podle následujícího vzorce:

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

DFT vezme na vstup vzorkovaný signál a na výstupu budeme mít obrazek signál rozložený ve frekvenční doméně. V praxi se ale používá jeho zrychlená podoba FFT (Fast Fourier Transformation), která má časovou složitost $\mathcal{O}(N \log N)$ oproti $\mathcal{O}(N^2)$ u DFT. Ukázka transformaci zvukového signálu do spectra pomocí DFT je na Obr. 2.3.



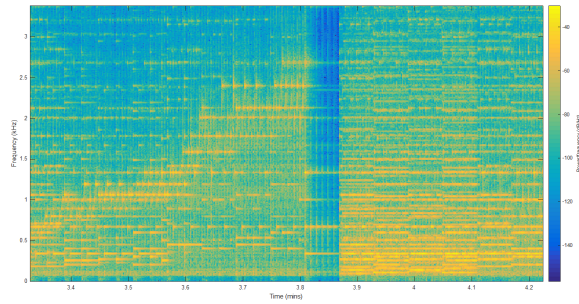
Obrázek 2.3: Příklad transformaci pomocí DFT

DFT (FFT) má ale jednu nepříjemnou vlastnost: když ji provedeme ztratíme informaci o čase. Například při transformaci hudební skladby pomocí FFT budeme vidět frekvenci a amplitudy z dané skladby, ale vůbec nebudeme tušit v jakém okamžiku tyto frekvence se objevili. Proto když chceme signál(zvuk) analyzovat v čase potřebujeme trochu jinou reprezentaci, kterou je spektrogram. Spektrogram dostaneme tím způsobem, že vstupní signál nasekáme na rámce (vynásobením signál posuvným oknem) a v každém rámci spočítáme FFT. Příklad spektrogramu je vidět na Obr. 2.4[?]

Tady je problém v tom, že přesnost v časové doméně a ve frekvenční jsou navzájem vylučující. Když zvyšujeme rozlišení v čase (např. při použití malého okna), ztratíme přesnost frekvenčního rozlišení a naopak. Zvuková reprezentace pomocí lineárně uspořádaných frekvenci neodpovídá přesně tomu jak zvuk vnímá člověk, proto v praxi klasický spéct (spektrogram) není moc populární.

Jednou z možných alternativ je Mel-spektrogram. Daný způsob vytváření spektra je inspirován přírodou — způsobem jak člověk dokáže vnímat různé frekvence.² Mel-spektrogram je klasický spektrogram, který je komprimován ve frekvenční ose, a proto může být efektivnější ve své velikosti při zachování nejdůležitějších informací.[?] Mel transformaci se dá popsat teto formulí:

²Člověk lépe rozlišuje nízké frekvence. Se zvýšením frekvenci schopnost člověka jich rozlišit logaritmicky klesá.



Obrázek 2.4: Spektrogram klavírní skladby [?]

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

kde f je právě transformovaná frekvence. [?] Ukázka Měl transformace je na Obr. ??

Jinou možnou alternativou je Constant-Q (CQT) spektrogram. CQT spektrum poskytuje 2D 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 ale nutné zdůraznit, že výpočet CQT je o něco náročnější než výpočet klasického spektrogramu nebo Mel-spektrogramu. Transformací klasického spektrogramu do CQT spektrogramu popisuje vzorek

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

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

Další reprezentaci zvuku je tzv. cepstrum [?]. Jedná se o výsledek inverzní Fourierova Transformace. Podobně jako u spektrogramů, existuje také varianta Mel-transformace cepstra – Mel-cepstrum.

2.2 Modely pro transkripci zvuku

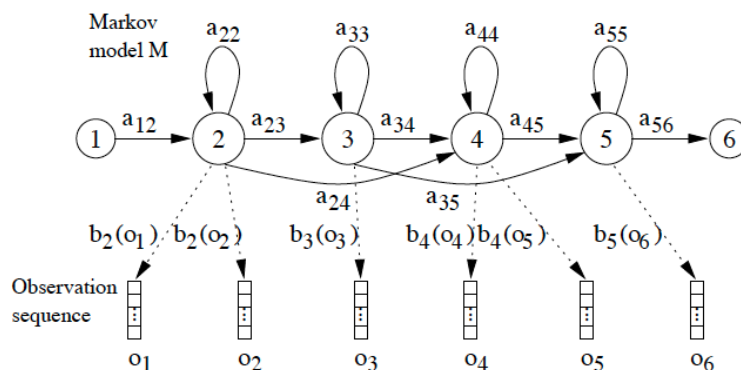
Pro účely klasifikace nebo rozpoznávání not se používají různé matematické modely. Mezi nejvýznamnější modely pro automatizovanou transkripci hudby patří skryté Markovské řetězce (hidden Markov chain, HMM [3]) a neuronové sítě [?]. Nejprve v rychlosti popíšeme HMM.

Definice 1. Necht 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 pokud platí:

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

Jinými slovy HMM je statistický model, který simuluje činnost procesů 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. Například existuje nějaký klasifikační úkol pouze s jednou třídou, potom HMM může vypadat následovně:



Obrázek 2.5: Skrytý Markovův model který dokáže říct, zda nějaký vzorek patří pouze do jedné konkrétní třídy [?].

Jak je vidět, HMM se skládá z několika skrytých stavů. Každý stav po trénování má uvnitř Gaussovou funkci. Přejchod mezi stavy se provádí s určitou pravděpodobností (na obrázku a_{ij}). Na vstup (do stavu 1) se podává nějaká sekvence. Výstupem potom bude míra podobnosti vstupní sekvencí a sekvence které HMM se naučil. Tato míra podobnosti se určuje tak, že každý stav podle naučené Gaussově funkci se počítá pravděpodobnost že i -ty element vstupní sekvence je shodný s j -tým elementem naučených sekvencí. Na základě těchto dílčích pravděpodobností výstupní stav se potom spočítá celkovou pravděpodobnost že vstupní sekvence je shodná se naučenými sekvencemi. Při výpočtu může nastat problém když vstupní sekvence má jiný počet elementů než počet stavů HMM ($I \neq J$). Tento problém se dá řešit sestavením různých cest pomocí Dynamic Time Warping metodu (popsáno dále). Když je potřeba rozpoznávat více tříd (což je typické) potom k tomuto modelu se přidává stejné modely (které ale se snaží naučit jiné vzorky). Vstupní vzorek je potom vzorkem třídy jejíž model má na výstupu nejvyšší pravděpodobnost.

I když HMM je velmi dobrý model pro klasifikaci audio nahrávek, pro porovnání s neuronovými sítí prohrává. Proto moje řešení bude založeno převážně na neuronových sítích. Jimž bude věnovat zbytek této sekce.

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

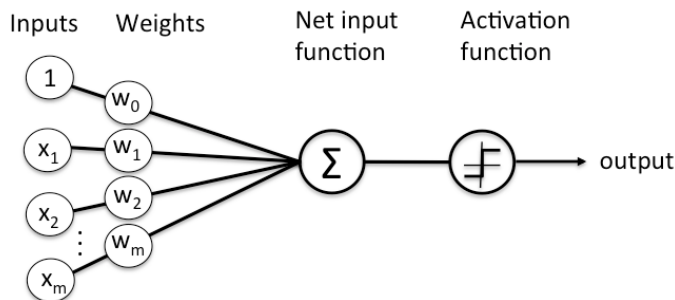
2.2.1 Neuron

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 obvykle se nazývá *aktivační funkce*. Jinými slovy se na začatek spočítá u (vzorek 2.2). Potom se přidává na

vstup určité aktivační funkci $f()$.

$$u = \sum_{i=0}^n w_i x_i, \quad (2.2)$$

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. Na Obr. 2.6 je vidět model umělého neuronu – Perceptron **TODO cite**. Jedna se ale o nejjednodušší verzi umělého neuronu. V praxi se občas používají i složitější modely.



Obrázek 2.6: Model umělého neuronu [?]

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 konvergence³, nebo v některých případech mohou aktivační funkce zabránit neuronovým sítím v konvergování.[?]

Aktivační funkce mohou být:

- Skokové (binární):

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

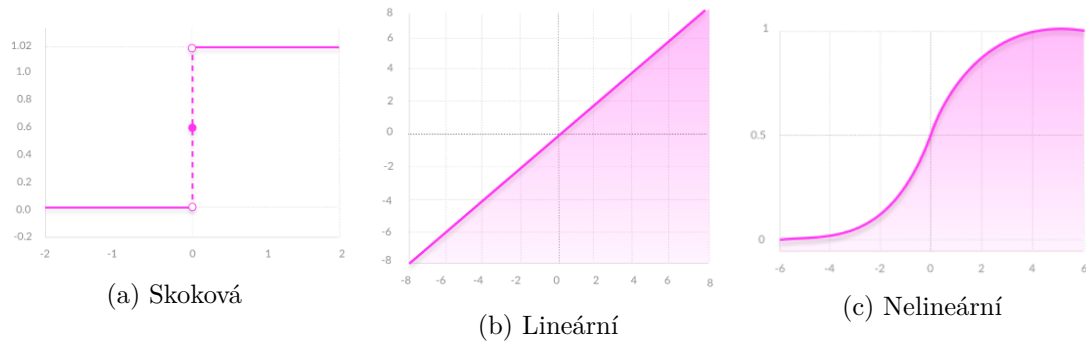
- Lineární: $y = ku$
- Nelineární: Např. $y = \tanh(u)$

Grafický průběh různých typu aktivačních funkcí je znázorněn na Obr. 2.7.

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, které jsou nelineární a zároveň mají vysokou dimenzi[?]

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. Dál budou popsány nejdůležitější aktivační funkce které se používají v klasifikačních úlohách.

³Konvergenci se tadý rozumí přibližování ke správnému řešení



Obrázek 2.7: Průběh různých aktivačních funkcí [?].

Sigmoida (obrázek vpravo 2.7) je klasickou aktivační funkcí která byla dříve moc populární a je vhodná pro všechny vrstvy. Má rozsah hodnot od (0, 1) a centrum v 0.5. Matematický je popsán vzorkem 2.3.

$$y = \frac{1}{1 + e^{-z}}, \quad (2.3)$$

Hyperbolický tangens (dal tanh) je další populární v minulosti aktivační funkce. Má rozsah hodnot od (-1, 1) a centrum v 0. Kvůli tomu má o něco lepší rychlost konvergence než sigmoida. Matematický je popsán vzorkem 2.4.

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad (2.4)$$

Nevýhodou tanh a sigmoidy je poměr — (výpočetní složitost/přínos). Sami o sobě tyto funkce velmi užitečné, ale teď existují více efektivní alternativy. Navíc tyto funkce jsou velmi citlivé na problém mizejícího gradientu⁴.

Průlomem stal vynález funkce ReLU (Rectified Linear Unit). Matematický tato funkce je popsána vzorkem 2.5.

$$y = \max(0, u). \quad (2.5)$$

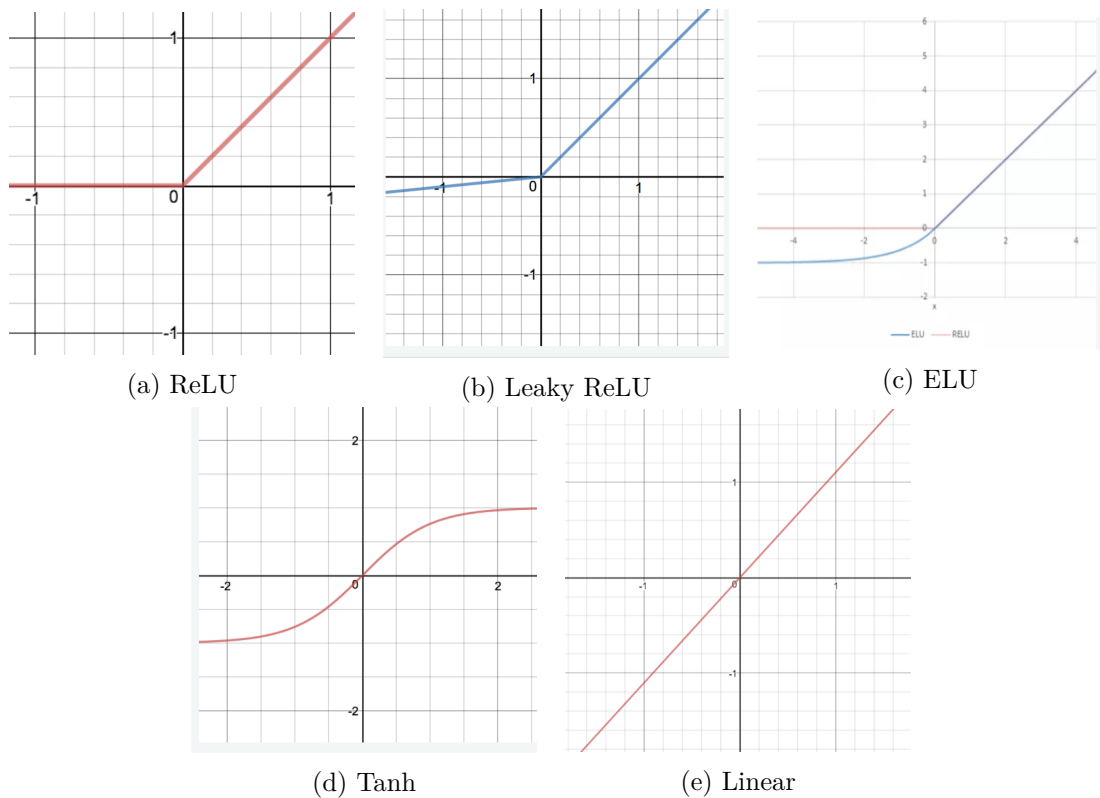
Kvůli své jednoduchosti, učení neuronu kteří mají uvnitř ReLU je několikrát rychlejší než učení tanh nebo sigmoidy. Ale ReLU neurony mají problém když na vstupu mají 0. V tomto případě klasický ReLU neuron ‘zemře’ (jeho výstupem od tohoto okamžiku vždycky bude 0). Což způsobí snížení rozlišovací schopnosti sítě. To se dá řešit pomocí drobných modifikací jako například Leaky ReLU.

Leaky ReLU — je modifikací ReLU funkce, která je popsána následující formou.

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

Když Leaky ReLU dostane na vstup zápornou nebo nulovou hodnotu, na rozdíl od Klasického ReLU vrátí malou zápornou hodnotu. A právě tímto zabrání ‘zemření’.

⁴Vanishing gradient problém — velmi důležitý problém při učení neuronových sítí. Při zvětšení počtu skrytých vrstev schopnost prvních vrstev se učit rychle stoupa dolů. To je dáno specifikou učení neuronových sítí a konkrétně BGD (Backpropagation Gradient Descent) algoritmem (<https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>).



Obrázek 2.8: Průběh aktivačních funkcí.

ELU (Exponential Linear Unit) — je silnou alternativou Leaky ReLU. ELU se dá popsat následujícím vzorkem.

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

kde α je předem stanovena konstanta. Dana aktivační funkce také umí produkovat záporné hodnoty. Nevýhodou je ale pro vstupní hodnoty > 0 může "vybuchnout" (dat na výstup příliš velké hodnoty).

Softmax — jedna se o specifickou aktivační funkci která přímo určena pro výstupní vrstvu. Má rozsah výstupních hodnot $(0, 1)$, přitom suma výstupních hodnot všech výstupních neuronů se rovna 1. To znamená že síť která na výstupu má Softmax aktivační funkci může specifikovat na kolik procent je jista že na vstupu vidí určitou třídu. Právě proto teď Softmax je standardem pro klasifikační úlohy. Dole 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 neuronu. Graficky znázornění průběhu aktivací funkcí je vidět na Obr. 2.8.

2.2.2 Učení neuronových sítí

Učením neuronových sítí se označuje proces během kterého se určují váhy (typický) každého neuronu při procházení trénovací sady. Na začátku učení jsou váhy inicializovaný malými

číslí kolem 0. Při zpracování každého vzorku se váhy sítě trochu mění. Epochou se rozumí počet iterací trénování pokud neuronová síť zpracovává každý vstup z datové sady právě jednou. Trénovací proces typický končí když přesnost neuronové sítě dosáhne předem nastavené konstanty anebo přesnost přestane se zlepšovat určitý počet epoch. Můžeme rozlišit několik přístupů k učení neuronových sítí.

Učení s učitelem. Je to druh učení, který je typický pro problémy jako regrese a klasifikace. Zde trénovací sada obsahuje dvojici (x, y) kde x je vstup pro neuronovou síť a y je požadovaný výstup neuronové sítě. Dostav na vstup x , síť produkuje vlastní výstup y' . Při porovnání y' a y trénovací proces dostává informaci v jakém směru musí upravovat váhy každého neuronu. Porovnání y' a y a extrakce informací o směru a velikosti chyb se provádí pomocí funkce ztrát (více dále).

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

Učení s podpořením Daný typ učení se používá když 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 ale může provádět experimenty — nějakým způsobem ovlivňovat prostředí a sbírat odezvy⁵ od prostředí. Právě na základě těchto dat neuronová síť se snaží naučit dělat správnou předpověď - volit nejlepší následující akci tak, aby v dlouhodobé perspektivě model dosáhl předem stanovenou cíle(stavu v systému). Tento typ učení obvykle se používá pro naučení správně strategie umělého agenta(robota, hráče ve hrách atd.).

Transfer learning Jedná se o metodu kterou se nedá zařadit do výše uvedených kategorií. Tato metoda může být aplikovaná ke každé z nich. Tato technika učení používá se když 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 tuto síť trénovat na vlastní datové sadě. Když síť byla pečlivě naučena na nějaké velké datové sadě, potom ona obsahuje uvnitř všechně potřebné nízké úrovně paterny, které je potřeba extrahovat ze vstupu. Nám zbývá jenom naučit poslední vrstvy tak, aby síť byla schopna použít tyto paterny pro náš specifický úkol.

2.2.3 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 pak dokáže změnit váhy tak, aby síť se o něco zlepšila. Existuje hodně druhů ztrátových funkcí které jsou účinné pouze pro malý seznam úloh. Proto výběr ztrátové funkce je velmi důležitý.

Obecně jich se dá rozdělit na dvě skupiny podle druhu úloh na kterých se uplatňuje model(neuronová síť):

- Regresní

⁵Odezvou se tady rozumí nový stav prostředí

- Klasifikační

Dana diplomová práce se zaměřená na klasifikaci vstupní nahrávky. Tady je těžko použit modely které jsou přímo zaměřeny na regresní úlohy. Proto za účelem dodržení rozměru diplomové práce dál budou popsány pouze aktivační funkce které jsou důležité pro klasifikační úlohy.

Mean Square Error (MSE) je klasickou funkcí ztrát ve strojovém učení. Dole můžete 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 to, 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 MSE a ostatní podobné ztrátové funkce (Mean Bias Error, Mean Absolute Error) se používají v regresních úlohách.

Binary Cross Entropy (BCE) je možnou alternativou k MSE. Obvykle se používá pro úlohy binární klasifikaci. Dá se 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 chybový a za příliš pochybující výstup. Používá se v klasifikačních úlohách, kde vstupní vektor se dá zařadit pouze do jedné třídy.

Co když nějaký vstupní vektor je potřeba zařadit do více tříd? Potom se jedná o multi-label klasifikační úlohu. V takovém případě se používá Categorical Cross Entropy. Dana 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 vektor⁶ p_{ij} je pravděpodobnost že vzorek i patří do třídy j .

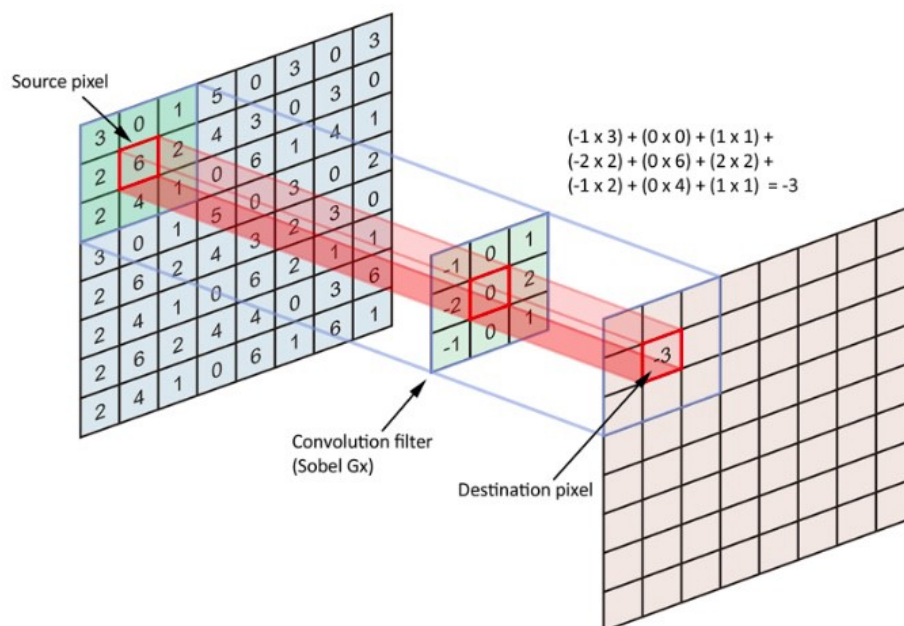
2.2.4 Vrstvy neuronových sítí

Jak bylo uvedeno dříve vrstvy představují skupinu podobných neuronů. Některé vrstvy jsou univerzální - dá se jich použít skoro v každé architektuře, některé jsou specifické - používají se pouze v specifických architekturách. Dana sekce bude popisovat nejčastější vrstvy neuronových sítí, které se používají v architekturách důležitých pro klasifikační úlohy.

Dense (plně propojená) vrstva Klasická vrstva kde každý neuron propojen s každým výstupním neuronem předchozí vrstvy. Typický 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.

⁶One-hot vektor, je vektor čísel 0,1. Délka vektoru se rovná počtu tříd a každé číslo říká zda příslušný vzorek patří do určité třídy

Konvoluční vrstva. Konvoluční vrstva představuje množinu neuronů které jsou odlišné od klasického chápání neuronu. Na rozdíl od Dense vrstvy neuron v konvoluční vrstvě připojen ("vidí") pouze k malému lokálnímu poli z předchozí vrstvy a během výpočtu prochází svým lokálním oknem celou předchozí vrstvou. Konvoluční neuron reprezentován konvolučním jádrem (matici). Při učení síť se právě snaží naučit správné hodnoty této matice. Pro získání výstupu neuronu, jeho konvoluční jádro se pohybuje nad vstupním obrázkem (nebo nad výstupem předchozí vrstvy) a vypočítá lineární kombinaci hodnot pod jádrem, kde jako koeficienty lineární kombinaci používá hodnoty v jádře. Na obrázku 2.9 můžete vidět ukázkou procházení konvolučního jádra.

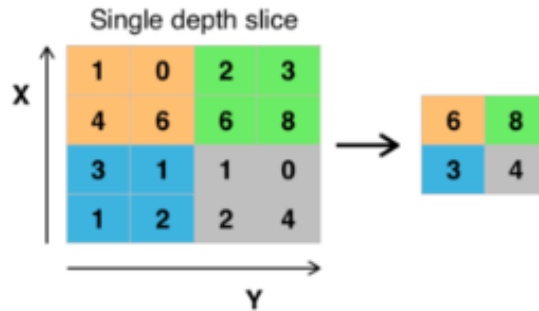


Obrázek 2.9: Výpočet výstupu konvolučního neuronu pro jednu polohu [?].

Tento typ vrstvy zpravidla reaguje na nějakou předem naučenou vlastnost: caru, hranu, kolo. Na základě nasbíraných vlastností plně propojené vrstvy potom dokážou velice snadno klasifikovat obrázek.

Sdružující vrstva Takže 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 takový podrobný obrázek již není zapotřebí pro další zpracování a je zkomprimován do méně podrobného. Kromě toho filtrování již nepotřebných částí pomáhá zabránit přeučení neuronové sítě. Sdružující vrstva se obvykle vkládá za konvoluční vrstvu před další konvoluční vrstvou. Na obrázku 2.10 je vidět výpočet pooling vrstvy pro jeden vstup.

Kromě sdružování s maximální funkcí můžete použít i další funkce - například průměrnou hodnotu nebo nějakou normalizaci.

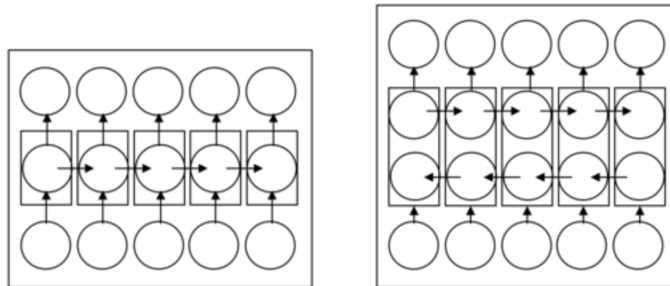


Obrázek 2.10: Komprimace 16 výstupu předchozí vrstvy do 4, [?].

LSTM LSTM vrstva je vrstva, která se skládá z LSTM modulů. Modul LSTM je modul pro cyklické sítě, který může ukládat hodnoty jak pro krátkou, tak pro dlouhou dobu. Na vstup bere vzorky které uspořádané v čase v jednom směru ($t_1, t_2, t_3 \dots$). Závazné je, že modul LSTM nepoužívá aktivační funkci uvnitř svých rekurzivních komponent. A tímto způsobem Rekurentní neuronové sítě(víc dal) používající LSTM jednotky částečně řeší problém mizejícího gradientu, protože LSTM jednotky umožňují gradientu průtok beze změny. Síť LSTM však stále mohou trpět explodujícím gradientem problému.[26]

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 (t_1, t_2, t_3 a zároveň t_3, t_2, t_1). [27]

Schematické zobrazení LSTM a BiLSTM sítí je vidět na obrázku 2.11.



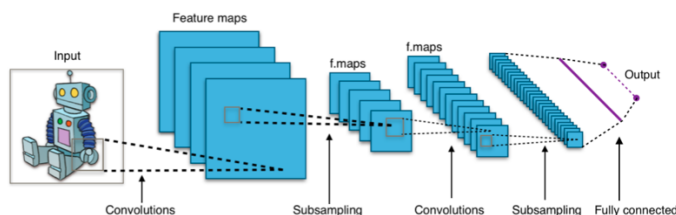
Obrázek 2.11: Schematické znázornění architektury LSTM (vlevo) a BiLSTM (vpravo) vrstev [?].

2.2.5 Architektury neuronových sítí

Existuje řada architektur které jsou zaměřeny pro různé účely. Následující sekce bude popisovat nejdůležitější architektury neuronových sítí vztahených ke klasifikaci signálu (včetně audia).

Konvoluční neuronové sítě To je 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 rozdílem oproti konkurenci. A od té doby to byl standard v klasifikaci obrázků. Myšlenka konvolučních neuronových

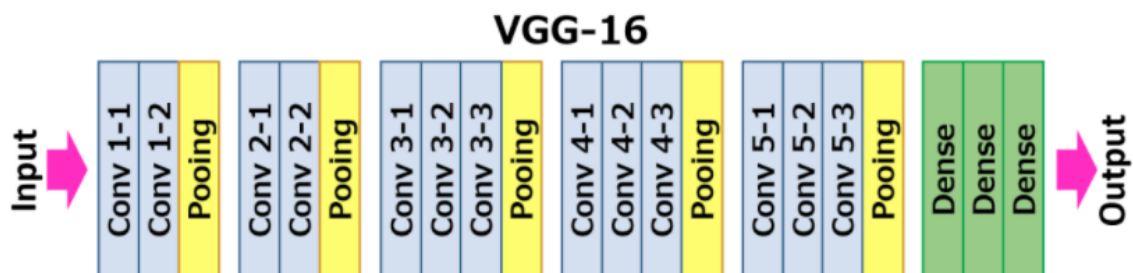
vých sítí je tedy střídat konvoluční vrstvy a sdružující vrstvy. Struktura sítě je jednosměrná (bez zpětné vazby). Klasickou strukturu neuronové sítě můžete vidět na Obr. ??.



Obrázek 2.12: Typická architektura konvoluční neuronové sítě [?]

Důležité reprezentanty V tomto segmentu budou popsány důležité reprezentanty konvolučních neuronových sítí. To jsou nejmenší a nejrychlejší neuronové sítě mezi konvolučními neuronovými sítí. Přesnost uvedených dále sítí je stále dobrá a právě proto ony začleněny do dané prací.

VGG16 VGG16 je konvoluční neuronová síť navržená K. Simonyanem et. al. (TODO VGG16odkaz), což je vylepšená verze AlexNet⁷. 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 parametru ale se zvýšil na 138 milionu. Při testování na ImageNet⁸ VGG16 dosahuje přesnosti 92,7 procenta⁹. Architekturu VGG16 můžete vidět na Obr. 2.13.



Obrázek 2.13: Architektura VGG16 kde:

Conv - konvoluční vrstva, Pooling - Sdružující vrstva a Dense - plně propojena vrstva.

MobileNetV2 MobileNetV2 je to konvoluční neuronová síť od Google, která je náhradou MobileNetV1. Tyto sítě mají malou vypočetní složitost (pouze 3.4 milionu trénovacích parametru) a proto primárně se používají na mobilních zařízeních. Architekturu dané sítě můžete vidět na Obr. 2.14

⁷AlexNet je konvoluční neuronová síť. Byla představena na konkurzu po rozpoznávání objektů v obraze v roce 2012. Od tohoto momentu konvoluční sítě se považují jako standart v klasifikaci obrazů.

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

⁹Sít VGG16 byla trénovaná několik týdnů pomocí grafických karet NVIDIA TITAN BLACK. Přitom měření přesnosti na ImageNet dělá takovým způsobem, že výsledek je počítán jako správný, pokud neuronová síť zařadí správnou odpověď mezi 5ti nejpravděpodobnějšími objekty na obrázku.

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×	Conv dw / s1	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 512$
	Conv dw / s2	$3 \times 3 \times 512$ dw
	Conv / s1	$1 \times 1 \times 512 \times 1024$
	Conv dw / s2	$3 \times 3 \times 1024$ dw
	Conv / s1	$1 \times 1 \times 1024 \times 1024$
	Avg Pool / s1	Pool 7×7
	FC / s1	1024×1000
	Softmax / s1	Classifier

Obrázek 2.14: Architektura MobilenetV2 kde:

Conv - konvoluční vrstva, Conv dw - Hloubkově oddělitelná konvoluce, Avg Pooling - Sdrůžující vrstva která redukuje na základě funkce *average()*, Dense - plně propojena vrstva.

V MobileNetV2 se používá nové typy vrstev, a to jsou:

- Relu6
- Hloubkově oddělitelná konvoluce

Relu6 je drobnou modifikaci klasické ReLu funkce. Jediným odlišením je to že maximální výstupní hodnota Relu6 může být 6. Nový matematický zápis této funkce je dole.

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

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

Hloubkově oddělitelná konvoluce (Depthwise Separable Convolution) je nová (relativně komplikovaná) modifikace konvoluční vrstvy. Hlavní myšlenkou této vrstvy je:

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

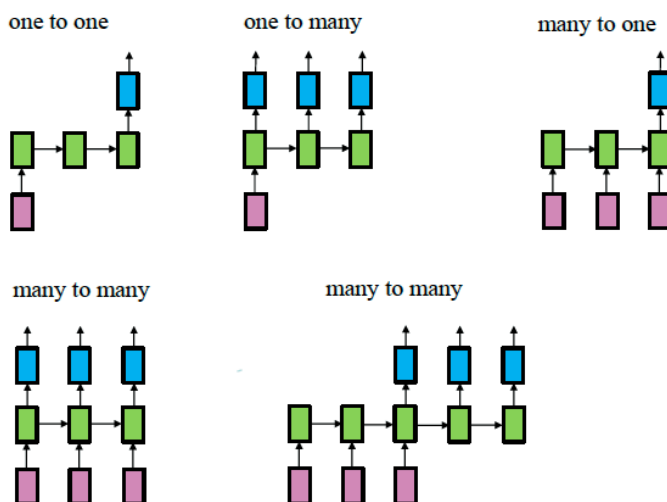
Použití této vrstvy dokáže zmenšit počet výpočtů, které je potřeba provést pro inferenci¹⁰ neuronové sítě. Zmenšení počtu výpočtů závisí na parametrech sítě a typicky výpočtů o 8-9 krát méně. Přesnost při tom klesá pouze trochu.

Rekurentní neuronové sítě Rekurentní neuronové sítě (RNS, RNN) je typ neuronových sítí, kteří primárně určeny k práci s časově závislými sekvencemi (řeč, písmo, hudba). Na rozdíl od vícevrstevných acyklických neuronových sítí RNN mohou 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 nejrozšířenější sít s dlouhodobou a krátkodobou pamětí (LSTM).[?, ?]

Obecně rekurentní sítě se dá rozdělit na:

- 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) - dá se aplikovat pro klasifikaci videa na úrovni snímků.

Na obrázku 2.15 je přehled druhů rekurentních neuronových sítí.



Obrázek 2.15: Schematické rozdělení RNN

[?]

Klasická RNN ale má určité problémy. Teoreticky, ony mohou sledovat libovolné dlouhodobé závislosti ve vstupních sekvencích. Potom problém je v tom, že když trénujete klasickou RNN pomocí zpětného šíření chyby, gradienty, které se šíří zpět, se mohou "zmizet" (mohou konvergovat k nule) nebo „explodovat“ (konvergovat k nekonečnu).[26]

¹⁰Inferencem neuronové sítě se tady rozumí proces, ve kterém neuronová síť ze vstupu vypočítá vlastní výstup ??

2.2.6 Příprava trénovací sady

Modely strojového učení typicky jsou velmi závisle na velikosti a kvalitě trénovací sady (dal datasetu). Od velikosti závisí jak dobře model bude rozumět reálným datům. Od kvality datasetu závisí jak rychle model a trénovací algoritmus dokážou extrahovat užitečné znalosti z dat a převést jich do parametru modelu.

I přesto ze pořízení velkého datasetu je problém není moc triviální. V rámci diplomové práce tento problém je těžko ovlivnitelný. Od nás ale úplně závisí kvalita dat, kterými budeme krmit model (například neuronové síti). Právě proto následující sekce bude věnovaná zlepšení kvality datasetu.

V případě že se jedná o učení s učitelem (viz 2.2.2) je potřeba aby každému vstupu odpovídal správný výstup. V případě práci se zvukem (dataset má tvar dvojic - audio, MIDI) je potřeba mít dobře zarovnaný dataset: kousek audia musí obsahovat pouze noty, které uvedené v seznamu MIDI. Zarovnání se dá provést pomocí DTW (viz dále).

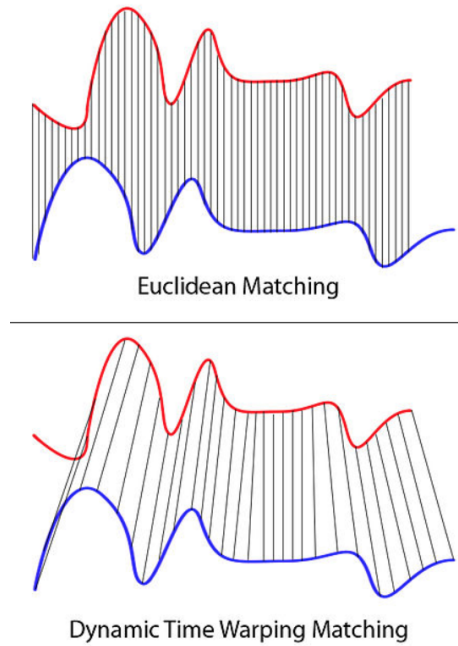
Je nezbytné mít v datech určitý úroveň šumu, aby model mohl po trénování porozumět skutečným datům. Ale úroveň šumu nesmí být matoucí pro model.

Pokud víme nějakou empirickou znalost o datech, tak je občas je dobrým řešením převést data do jiné reprezentaci, která zachovávala tuto znalost. A tím docílíme velkého redukování času, který je potřeba věnovat trénování modelu. Protože model už nebude nucen naučit již zmíněnou znalost. V případě že se jedná o audio nahrávku, a chceme tuto audio nahrávku klasifikovat, nemusíme model krmit PCM vzorky (viz 2.1). Model mnohem jednoduše dokáže klasifikovat nahrávku, pokud dostane na vstup místo PCM vzorků nějakou frekvenční reprezentaci této nahrávky. Také algoritmy strojového učení jsou velmi závisle 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 aktivní funkce v případě neuronových sítí). Například pro SVM klasifikátor nejlépe vyhovuje rozsah vstupních hodnot $[-1,1]$ a směrodatnou odchylku 0.5 s centrem v 0.

Dynamic Time Warping (DTW) To je algoritmus pro nalezení míry podoby mezi dvěma sekvencemi s různou délkou. Hlavní problém s porovnáním takových sekvencí je to že jejich porovnání Eukleidovskou vzdáleností často selhává. To je dáno tím, že Eukleidovská vzdálenost se vypočítá postupným porovnáním prvků se stejnými indexy. Potom v situaci, když budou porovnané dvě skoro stejné sekvence (vektory), ale jeden z nich bude posunut o jeden prvek, Eukleidovská vzdálenost může považovat tyto vektory za úplně odlišné. Na rozdíl od Eukleidovské vzdálenosti DTW se snaží co nejlépe namapovat prvky (indexy) jednoho vektoru na druhý. Potom nejlepší mapování se považuje za délku (rozdíl) mezi vektory. Rozdíl mezi eukleidovskou metrikou a DTW je na obrázku 2.16.

Nechť máme 2 vektory r a t s délkami k a n , potom zarovnání probíhá následujícím způsobem [?]:

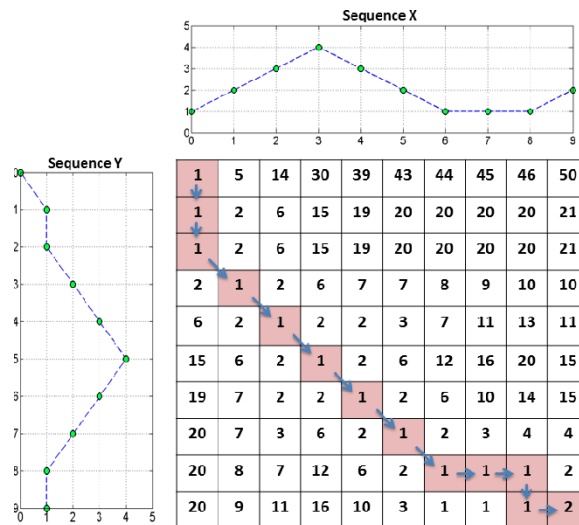
1. Sestavena matice M kde každý prvek $M(i, j)$ je dan výpočtem distance mezi vektory $r(i)$ a $t(j)$.
2. Probíhá výpočet cesty mezi prvkem $M(0, 0)$ a $M(k, n)$.
3. Buňky matice které jsou patřící k určité cestě se sčítají. A tím dostáváme kumulovanou vzdálenost dvou vektorů.
4. Tato kumulovaná vzdálenost potom může být normalizovaná.
5. Nejmenší vzdálenost potom považována za distanci mezi vektory r a t .



Obrázek 2.16: Rozdíl při porovnání pomocí euklidovské metriky (nahore) a DTW (dole) [?]

6. Průběh výpočtu můžete vidět na obrázku dole.

Grafické znázornění porovnání dvou sekvencí pomocí DTW je na obrázku 2.17.



Obrázek 2.17: Výpočet DTW [?].)

2.2.7 Metriky trénování

K tomu abychom pochopili zda nějaká neuronová síť se zlepšuje či ne, musíme zavést určité metriky, které nam budou tuto informaci poskytovat. Takových metrik existuje celá řada. Neda se zvolit pouze jednu, protože různé metriky poskytují různé vlastnosti kva-

lity neuronové sítě. V dalším textu budou popsány nejdůležitější metriky pro trénování neuronových sítí.

Předtím, než vysvětlovat různé metriky je potřeba zavést pojem *chybové matice*. Chybová matice je to specifická tabulka, která umožňuje vizualizovat výkon algoritmu (typický 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 můžete vidět v tabulce 2.1.

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

Tabulka 2.1: Chybová matice pro dvě třídy, kde: y' - je predikce modelu, y - je skutečná hodnota reálného světa.

Z ní dostáváme 4 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 odpovědi vůči celkovému počtu odpovědi. Vypočítá se podle vzorce 2.6:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.6)$$

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

Na rozdíl od *accuracy*, *precision* a *recall* není závislé na poměru počtu prvků v třídách a proto jejich se dá použít i pro nevyvážená trénovací data. Dá se jich počítat podle vzorků 2.7 a 2.8

$$precision = \frac{TP}{TP + FP}, \quad (2.7)$$

$$recall = \frac{TP}{TP + FN}. \quad (2.8)$$

Občas je potřeba popsat kvalitu neuronové sítě jednou hodnotou. K tomu se používá metrika *f1*. Jedná se o harmonický průměr *precision* a *recall*. Výpočet této metriky je popsán vzorcem 2.9.

$$f1 = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall}, \quad (2.9)$$

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

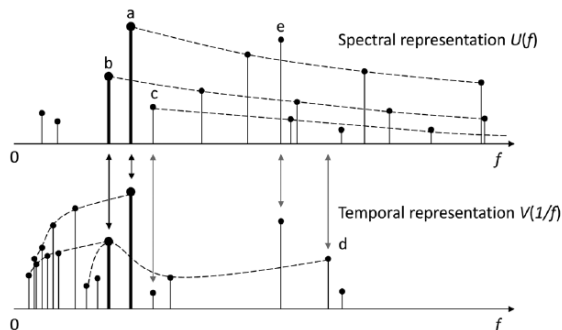
Taký často se sleduje metrika *Loss*. Jedná se o hodnotu ztrátové funkce, která byla určena pro danou neuronovou síť (více 2.2.3.).

2.3 Existující přístupy pro transkripci zvuku

Než se začneme navrhovat vlastní aplikaci je potřeba dobře prozkoumat již existující řešení. To nám umožní odhalit slepé uličky a přímo se pustit posouvat správnou technologii

dopředu. V této kapitole budou probrány různé způsoby klasifikací audio vzorků klavírní hudby, což je klíčovým krokem k transkripci audiosignálu do not.

Autoři [4] zároveň analyzovali log-scaled spektrum a cepstrum za účelem lepšího výběru kandidátních not v určitém kusu kompozice. Obr. 2.18.



Obrázek 2.18: Log-scaled spektrum a cepstrum [?].

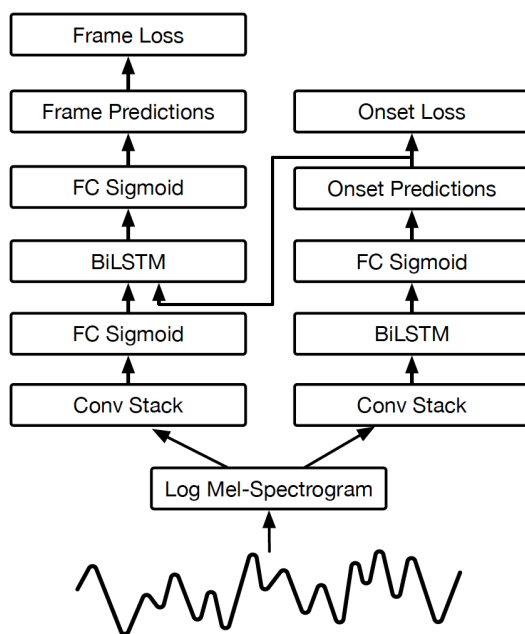
Potom na to se uplatňovali klasické metody zpracování signálu. Jako výsledek - daný přístup ukazuje lepší hodnoty přesnosti pouze na jednom datasetu z sedmi. Ale stále je to byl dobrý alternativní pokus bez použití neuronových sítí.

V [] autoři použili HMM (viz 1) pro klasifikaci audio vzorku za účelem odhalení plagiátu v nové vymyšlených kompozicích. Navržený model dosahuje průměrné přesnosti skrz všechny typy datasetu 67 procent, což je relativně málo. Natolik mála přesnost podle mě způsobena příliš malým datasetem kterým disponovali autoři.

Práce [5] používá neuronové sítě a založena na myšlence že některé rámce jsou důležitější než ostatní. Tyto rámce jsou rámce s začátkem noty (protože hned po začátku znění energie noty se začne klesat). A právě v těchto rámcích máme co nejvíc informací pro klasifikaci vzorků. Za účelem nalezení těchto rámců byl natrénován zvláštní detektor začátku noty založený na konvolučních a rekurentních vrstvách. Vstupem pro tento detektor je Mel-spektrogram. Jehož výstupem je 88 (klavír má právě 88 kláves) pravděpodobnosti že určitá nota se právě zazněla v tomto rámci. Tento výstup je přímo použit jako doplňkový vstup pro samotný detektor not. Celková architektura sítě je zobrazen níž na Obr. 2.19.

Metoda navrhovaná v [6] je založena na metodě představené v [5]. Neuronová síť (model) byl převzat pouze z drobnými úpravami, které se týkali pouze zvýšení počtu neuronů ve vrstvách. Závazným rozdílem [?] od [?] je ale to, že autoři v (6) použili pro trénování modelu jiný dataset, čím dosáhly obrovského přírůstku přesnosti. Vzniklý dataset se jmenuje MAESTRO v1. Byl sestaven pomocí nahrávek vytvořených během 9-ti let pianových konkurzů. Obsahuje synchronizované na 3 ms (audio - MIDI) dvojice. Tak přesná synchronizace byla dosažena minimalizací distancí mezi CQT rámcem z audia a vygenerovaným audiem na základě příslušného MIDI. Minimalizace distancí bylo provedeno pomocí technologie DTW. V Tab. 2.2 můžete vidět porovnání různých datasetů s MAESTRO v1.

Je vidět, že MAESTRO v1 má skoro 10 krát větší počet not oproti MAPS datasetu (na němž byl trénován model z prací [5]).



Obrázek 2.19: Architektura sítě [?]

Dataset	Počet skladeb	Celková delka	Počet not(millin)
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 2.2: Porovnání datasetů [?].

Kapitola 3

Navrh systému

Cílem této práce je navrhnout a implementovat systém, který by umožňoval uživateli cvičit libovolnou pianovou kompozici k níž má nahrávku. Systém má dvě klíčové funkce:

1. Převod nahrávky do note batches reprezentaci (viz dál).
2. Během cvičícího procesu kontrolovat v reálném čase co přesně hraje uživatel.

První bod — transformace nahrávky do note batches — se dá implementovat za použití neuronovou síti z prací [6]. Přesnost jejího modelu je těžko dosažitelná v rámci diplomové práce a proto tento model (dál velká neuronová síť) bude skoro bez změn zahrnut do mého systému. Nedostatkem modelu z (6) je malá rychlost, typický zpracování vzorků délkou x sekund trvá $x \times 2$ sekund. Což neumožní použití tohoto modelu ve druhém bodě. Proto jsem použil jinou neuronovou síť (dal real-time neuronová síť) jejíž cílem je rychle poskytnutí klasifikaci vstupních audio vzorků z mikrofonu.

3.1 Notova reprezentace — note batches

V průběhu cvičícího procesu je potřeba rychlé, bez zbytečných vypočtu generovat seznam not, které mají být vykresleny na obrazovku v následující okamžik. Z již existujících formátů zaměřených na notový zápis, MIDI vypadá jako nejvhodnější. Při použití MIDI je ale potřeba projít soubor v několika cyklech:

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

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

Tato notová reprezentace umožňuje rychle kreslit poznámky na obrazovce. Vše, co musíte 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

Fur Elise Clavierstück in A Minor - WoO 59

Ludwig Van Beethoven



Hudební zapis not

Obrázek 3.1: Ukázka různých způsobů reprezentaci not.

3.2 Cvičící proces

Cvičící proces obsahuje tři sub-procesy: řídicí proces, proces poslouchání a rozpoznávání. Každý sub-proces má cyklus který vykonává určitou hlavní činnost. Tyto činnosti budou popsány níže.

paragraphŘídicí proces

Z reprezentaci note batches aplikace bude postupně vykreslovat nahoru obrazovky noty, kde každá nota bude reprezentovaná obdélníkem. Během času aplikace bude postupně posouvat noty dolu dokud nějaká nota (obdélník) se nedotkne klávesy v dolní části obrazovky. Když nota se dotkla klávesy proces přestane posouvat noty dolu. Proces prodlouží posouvat noty dolu až zjistí že všechny potřebné klávesy byli zmáčknuty. Klávesy které byli zmáčknuty v tento okamžik proces dostane ze seznamu - *list_prediction*.

Proces poslouchání Tento proces pravidelně sbírá audio signal z mikrofону, uplatňuje na tento signal filtr pro redukující šumu (viz dal), a ukládá výsledný signal do seznamu - *list_listening*.

Proces rozpoznávání Tento proces bude spuštěn zároveň s předchozími procesy. Daný proces bude v cyklu:

1. Načítat audio vzorky ze seznamu - *list_listening*.
2. Převádět tyto audio vzorky na vstup do real-time neuronové síti.
3. Zapisovat výstup real-time neuronové síti do seznamu *list_prediction*.

3.3 Real-time neural network

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

3.3.1 Architektura neuronové síti

Od výběru vhodné architektury závisí rychlost výsledné neuronové síti a taky schopnost budoucí neuronové síti zobecnit data, která ta uvidí v datasetu. Od vyberu vhodné velikosti závisí schopnost neuronové síti dosáhnout požadované přesnosti. Proto je tady důležité zvolit správnou architekturu a zvažovanou velikost. Ale s rostoucí velikostí model se těžší trénuje. Kvůli tomu nejjednodušším řešením by bylo použít již trénovanou neuronovou síť na jiných datech a pomocí Transfer learningu (viz 2.2.2) přizpůsobit tuto síť k naší aplikaci. Jako kandidáty byli zvoleny VGG16 a MobileNetV2. Důvody proč právě tyto modely jsou následující:

- VGG16 je jednou z nejmenších konvolučních neuronovou síti, ale přesto přesnost této síti jenom o trochu hůř než u větších analogu.
- MobileNetV2 je konvoluční neuronová síť která primárně určena k běhu na mobilních zařízeních a proto musí být velmi rychlá.

Po změně počtu neuronů výstupní vrstvy na 88 (aby sítě byli užitečný pro klasifikaci not) byli provedeny testy rychlosti. Na vstup 10000 krát byl 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 sítí

Jak je vidět nejbližší výsledky k real-time ukazuje MobileNetV2. Ale přesto že minimální hodnoty úplně stačí pro účely výsledné aplikace, průměrná hodnota bude způsobovat citlivé zatížení systému. Tady je potřeba brát v potaz, že než neuronová síť začne zpracovávat vstup, na začátek je potřeba načíst z mikrofону audio vzorky(20ms) a provést redukcí sumu a převést výsledný signál do CQT(víc dal). Právě proto rychlost uvedených výše neuronových sítí nestačí.

3.3.2 Vlastní neuronová síť

Z uvedených výše důvodu bylo rozhodnuto trénovat vlastní neuronovou síť úplně od začátku. Jako možné kandidáty architektur se dá považovat RNN a CNN. Pomocí frameworku Keras jsem vytvořil malé neuronové sítě uvedených typu. A na málem datasetu jsem porovnal rychlosti s jakou dokážou trénovat. Bylo provedeno jedno malé trénování. Účelem daného trénování bylo se podívat jak uvedené architektury neuronových sítí zvladnou data z naše datové sady. Jako limit délky trénování byl stanoven na 100 epoch. Výsledky můžete vidět v tabulce 3.2 Jak je vidět přesnosti sítí jsou srovnatelné, ale délka trénování u RNN mnohem

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 epocham trénování

výše. Proto k dalšímu trénování byla zvolena pouze CNN síť. Výsledky testu rychlosti pro CNN jsou popsány v tabulce 3.1 v řádce *Real-time NN*.

3.3.3 Vytváření datasetu

K tomu abych dokázal natrénovat vlastní neuronovou síť je potřeba vytvořit dataset. Autoři (6) po trénování jejich neuronové sítí pokračovali ve sbírání dat do nového datasetu -

MAESTRO v2, který byl zveřejněn na podzimu 2019. Tento dataset obsahuje 11 krát více dat pro trénování než předchozí dataset MARS (více porovnání datasetu 2.2) a má celkem 121.8 GB pianových nahrávek a příslušných MIDI souborů.

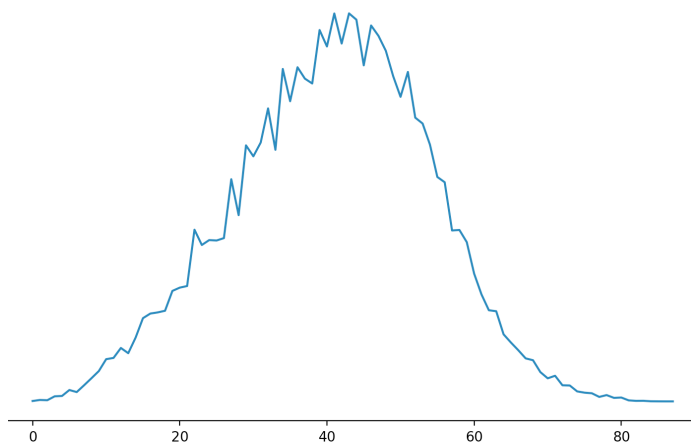
paragraphPříprava datasetu

Původně dataset MAESTRO v2 obsahuje *.wav* kompozice a příslušné *.MIDI* soubory. V takové podobě dataset je skoro neužitečný. Proto musíme jeho určitým způsobem upravit tak, aby bylo možné na něm trénovat naši neuronovou síť. Úpravy budou se tykat:

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

paragraphČištění dat Po zkoumání 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 tyto odlišné nástroje by pouze zvětšovali šum a bránili by neuronové síti zobecnit užitečná data. Proto každý audiozáznam (*.wav*) v datasetu byl v rychlosti poslechnout a kompozice s více nástroji byli eliminovány z datasetu. Celkem bylo eliminováno 6 kompozic.

Taky v datasetu byla odhalena velká disproporce not. Na Obr. 3.2 se ukazuje počet každé noty v datasetu.



Obrázek 3.2: Distribuce not v původním datasetu

Jak je vidět z obrázku počet vzorků s noty od 25 do 60 je mnohem vyšší než ostatních. Tato skutečnost může způsobit úplné ignorování not s malým počtem výskytu. Kvůli tomu a faktu že v rámci diplomové práce těžko trénovat neuronovou síť s > 100 GB dat, bylo rozhodnuto zmenšit dataset. Ale zmenšování bude probíhat takovým způsobem, který co nejvíce upraví distribuci počtu not v datasetu. V podstatě nebudu redukovat tento dataset, ale spíš vytvářím nový dataset ze starého.

Algoritmus pro vytváření nového datasetů:

1. Na začátku dataset je 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. Nad každou dvojici souboru (.wav, .MIDI) provést diskretizaci s krokem v 20 ms. Z toho dostaneme dvojici (PCM vzorky, seznam MIDI kódu).
4. Eliminovat zbytečné dvojice(víc dal).
5. Přidat augmentované vzorky(víc dal).
6. Pokud nepřevyšeni limit ve počtu vzorků nebo v obsažené paměti pokračuj, jinak přeskoč na bod 8.
7. Pokračuj od 3.
8. Dataset je připraven.

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

- Seznam MIDI kódu je prázdný: dataset i bez těchto trénovacích dvojic obsahuje velkou množinu příkladů kdy nějaká nota je potichu.
- V této dvojici teprve začala nebo skončila znít nějaká nota: 20ms je zcela malé okno, ale klidně může zachytit změnu, což bude znamenat že část PCM vzorku neodpovídá seznamu MIDI kódu.
- Potom z našeho datasetu chceme extrahovat co nejvíc různých vzorků. Ale máme redukovat 120 GB dat do toho co bychom dokázali časově rozumně zpracovávat. Tento limit velikosti byl stanoven na 14 GB. Proto do výsledného datasetu byl zařazen pouze každý 5. vzorek.

Augmentace dvojic. Jak je bylo uvedeno předtím, máme příliš velkou disproporci počtu not v datasetu. Proto kdy se nám povede najít dobrý PCM vzorek který budou mít pouze noty které nám chybí, potom budeme chtít několikanásobně kopírovat tuto dvojici do našeho budoucího datasetu. Ale jednoduché kopírování taky může uškodit - neuronová síť bude podceňovat ostatní dvojice. Proto při kopírování trénovací dvojice která už je obsazena v datasetu, do PCM vzorku bude přidán malý aditivní bily sum¹.

Po provedení výše uvedeného algoritmu nad původním datasetu dostali jsem nový dataset. Distribuce počtu not nového datasetu je ukázaná na Obr. 3.3.

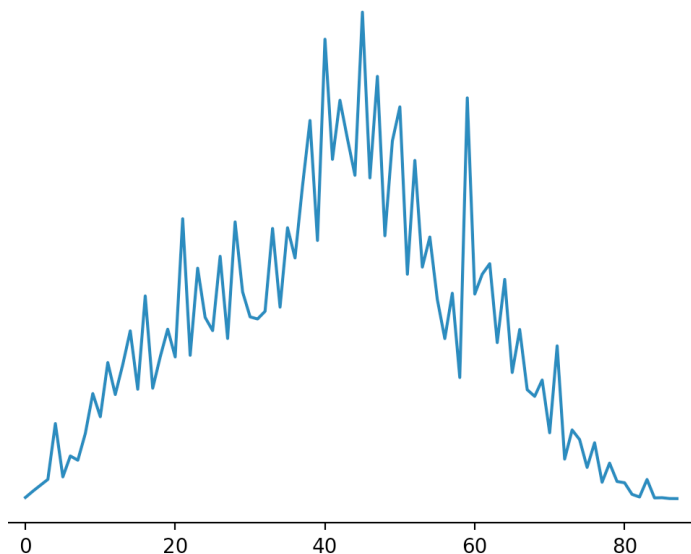
Jak je vidět, tato transformace dokázala aspoň trochu vyvážit distribuci počtu not.

Transformace dat. Když na dataset bude obsahovat dvojici (audio vzorky, odpověď) neuronová síť bude se učit velmi dlouho. Typický, za účelem pomoci síti zobecnit data, na audio vzorky se aplikuje nějaká z transformací: FFT, Mel-spec, CQT(vic 2.1.3). V případě že se jedná o klasifikaci zvuku obvykle se používá Mel-spec nebo CQT. K tomu abych zvolil druh transformace zase byla provedena série experimentů. Z datasetu který obsahoval trénovací dvojice (PCM vzorky, seznam MIDI kódu) bylo vytvořeno dva dalších datasetu:

- (Mel-spektrum, seznam MIDI kódu)
- (CQT-spektrum, seznam MIDI kódu)

CNN síť byla trénována 100 epoch pro každý druh datasetu. Výsledky experimentů jsou dole v tabulce 3.4: Jak je vidět z dané tabulky, výsledky nad CQT-datasetem mnohem lepší.

¹TODO Bily sum. numpy



Obrázek 3.3: Distribuce not v novem datasetu

Právě proto výsledný dataset bude tvořen pomocí CQT transformací audio signálu.

Jak bylo popsáno v teoretické části algoritmy strojového učení lépe pracují na normalizovaných datech - rozsah hodnot vstupních dat musí být $(-1, 1)$. Proto poté co jsme vytvořili CQT-dataset, tento dataset bude normalizován - CQT koeficienty budou převedeny do jiného měřítka (od -1 do 1).

Následně dataset byl rozdělen do třech menších datasetů: trénovací, testovací a validační. Trénovací dataset se používal pro trénování neuronové sítě. Na validačním datasetu byla provedena průběžná kontrola, zda nedošlo k přeučení neuronové sítě (více dále). Kontrola byla prováděna jednou za 5 epoch. Testovací dataset byl použit až na konci k hodnocení výsledné neuronové sítě. Počet trénovacích dvojic v každém datasetu je znázorněn v tabulce 3.3.

Typ datasetu	Počet trénovacích dvojic
Trenovací	1 722 685
Validační	215 335
Testovací	215 336

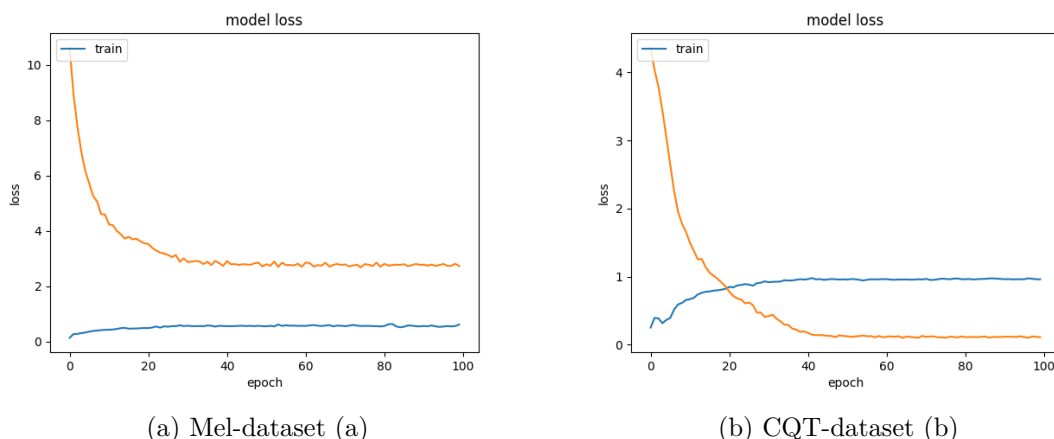
Tabulka 3.3: Rozdělení datasetu na tři dalších datasety.

3.4 Implementace

Tato sekce udělá rychlou rekapitulaci problému a řešení vzniklých během implementaci vyzvedne aplikaci.

3.4.1 Volba užitečného ML frameworku

Pro trénování neuronových sítí existuje spousta ML(strojové učení) frameworku. Nejrozšířenější jsou Keras a Pytorch. Pytorch je framework pro strojové učení od Facebook který je



Obrázek 3.4: Porovnání rychlosti trénování neuronové sítě nad CQT-datasetem a Mel-datasetem. Modrou barvou označena přesnost neuronové sítě, oranžovou barvou označena ztrátová funkce (více 2.2.3).

zaměřen spíše na zkoumání: dá se dynamicky měnit architekturu během trénování, ale inference² sítě prochází pomaleji než u Kerasu. Taky nevýhodou Pytorch je to, že on nemá implementované multiprocessingové trénování. Implementovat samostatně multiprocessingové trénování neuronové sítě může výrazně zkomplikovat vytváření aplikace. Proto jako trénovací framework byl zvolen Keras. Keras ale má taky výrazný nedostatek — když neuronová síť bude dělat inference modelu, Keras bude načítat celou neuronovou síť do operační paměti úplně znovu. Načítání typický zabere půl vteřiny. To bude negovat veškeré naše úsilí o výběr správné architektury. Řešením daného problému je převod Keras modelu do ONNX (Open Neural Network Exchange) modelu. ONNX je knihovna která se používá pro výměnu modelů mezi různými ML frameworky. Kromě toho ale ONNX má i vlastní inference. Tento inference nepotřebuje pokaždé načítat neuronovou síť do operační paměti (stačí pouze jednou, na začátku). Dany fakt výrazně zkrátí časovou zátěž výsledné aplikace.

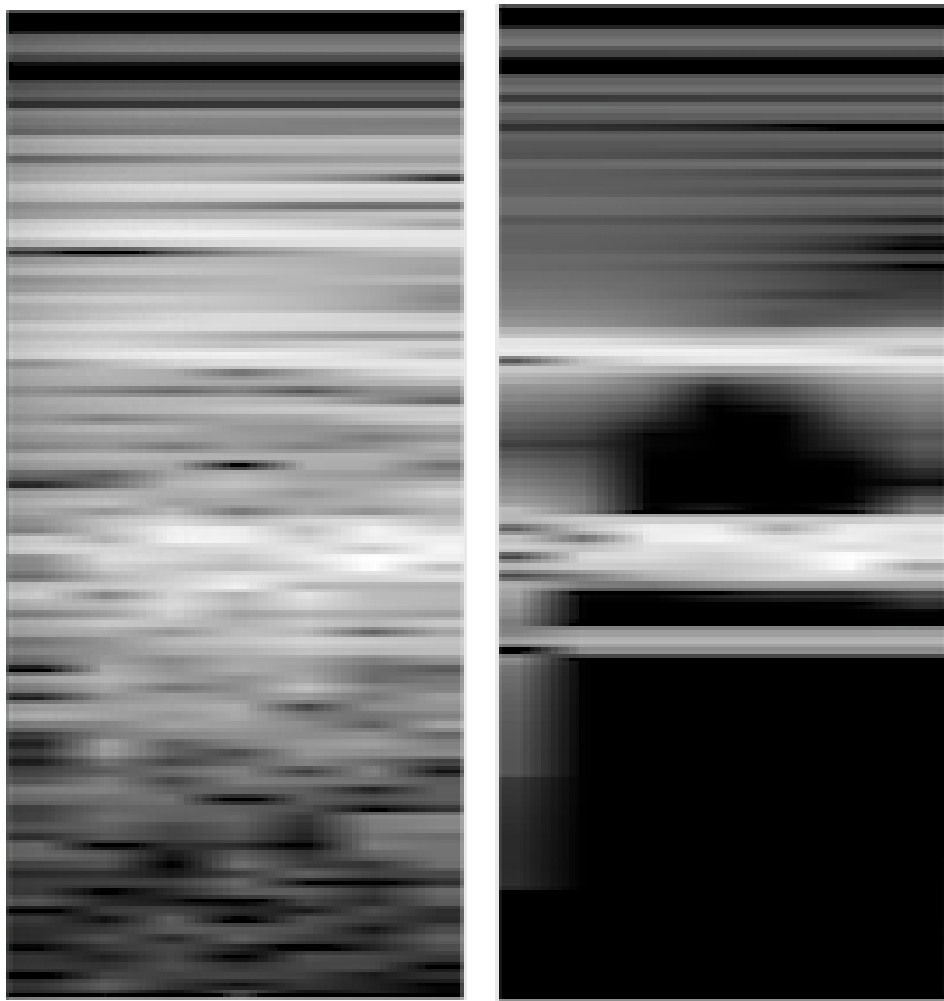
3.4.2 Předzpracování vstupního signálu, redukce sumu.

Tato aplikace je určena pro spuštění na vlastním počítači, nikoliv na profesionálním přístroji v studii. A proto existuje velké riziko najít ve vstupním signálu z mikrofonu hodně šumu. Vliv tohoto šumu je potřeba redukovat. Za tímto účelem v aplikaci je použita redukce šumu na základě FFT (viz 2.1.3). Algoritmus filtraci šumu je následující:

1. Převést vstupní vzorky do FFT.
2. Na základě pevně stanoveného prahu odstranit všechny frekvence které mají nízkou sílu.
3. Provedením inverzní FFT dostat audio signál s redukováným šumem zpátky.

Vliv použití daného filtru na reálný audio signál je vidět na 3.5.

²Inference je to část programu, která dokáže spustit neuronovou síť na základě souborů kde daná síť je zapsaná.



(a) Původní signál

(b) Filtrovaný signál

Obrázek 3.5: Porovnání spektrogramu původní nahrávky z mikrofonu (a) a nahrávky s redukováným šumem (b).

3.4.3 Trénování real-time neuronové sítě

Bylo vyzkoušeno několik architektur typu CNN. Jako nejvíc perspektivní se ukázala architektura která znázorněna v tabulce. 3.4 Trénování probíhalo na vypočetním klasteru

Typ vrstvy	Počet neuronů
Conv	100
Pooling	-
Conv	100
Pooling	-
Conv	100
Pooling	-
Dense	1000
Dense	512
Dense	88

Tabulka 3.4: Finalní architektura real-time neuronové sítě, kde *Conv* - konvoluční vrstva, *Pooling* - sdružující vrstva, *Dense* - plně propojena vrstva.

- METACENTRUM **TODO cite**. Výsledná síť byla trénována během 5ti dnů. Za tento čas bylo provedeno 97 epoch trénovacího cyklu. Za účelem zabránění přeučení bylo nastaveno dočasné ukončení trénování v případě, že ztrátová funkce nebude se zlepšovat během 10ti eopch. Kvůli tomu že nedošlo k dočasnému ukončení se dá říct, že síť má potencial se zlepšit při pokračování trénování v budoucnu.

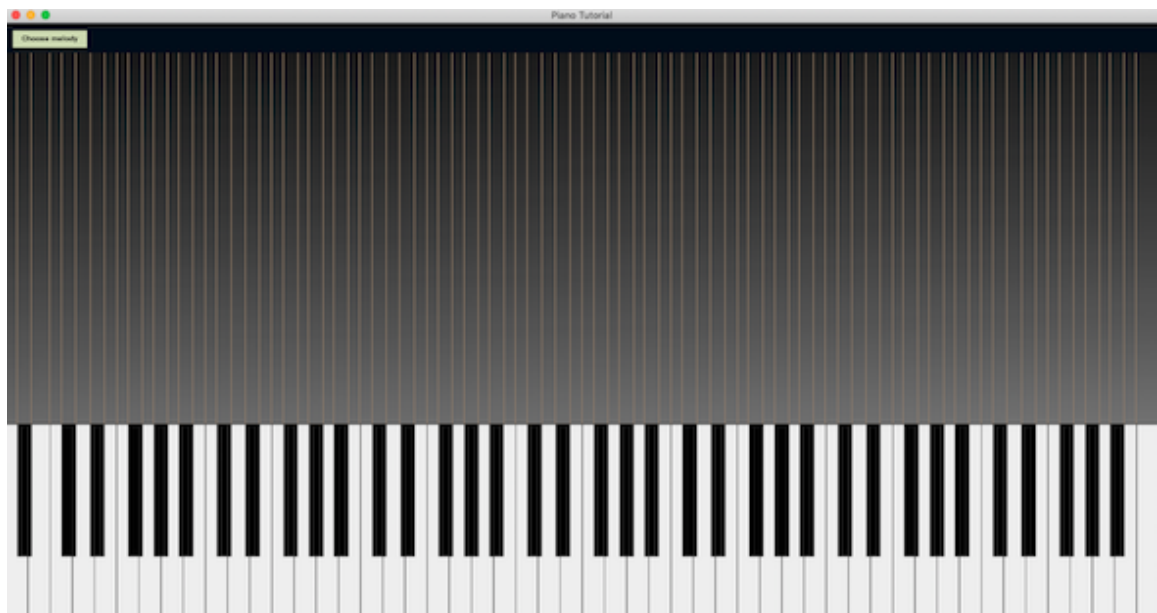
3.4.4 Grafická část

Grafická část je implementovaná s využitím knihovny Pygame. Z důvodu vzniklých problému ve již vyřešených modulech programu celková aplikace bude mít redukovanou funkcionalitu, a to za účelem robustního fungování přípustných modulu. Aplikace bude mít pouze jedno okno které je znázorněné na obrázku 3.6.

Okno obsahuje pouze jedno tlačítko — Výběr Souboru. Po zmačknutí na toto tlačítko uživateli se zobrazí obsah složky songs (budou zobrazeny pouze soubory: .midi, .mid, .wav a .csv). Pokud uživatel bude chtít cvičit kompozici z vlastního souboru, pak bude muset umístit vlastní soubor do této složky. Když uživatel zvolí soubor který se chce naučit, aplikace začne cvičicí proces. Když poslední obdélník (nota) zmizí, cvičicí proces se automaticky zakončí.

Uživatelský workflow se dá shrnout do následujících bodů:

1. Uživatel zvolí nahrávku pianové kompozici.
2. Aplikace na základě velké neuronové sítě převede tuto nahrávku do speciální vnitřní reprezentaci - note batches.
3. Aplikace na základě reprezentaci note batches začne cvičicí proces.



Obrázek 3.6: Ukazka GUI

Kapitola 4

Experimentální vychodnocení

Dana kapitola se zabývá experimentálním ohodnocením vytvořené aplikaci.

4.1 Časové testy

V tabulce 4.1 jsou znázorněny časy běhu jednotlivých modulů. Tabulka taky obsahuje komulované časy které udávají přehled o celkové rychlosti aplikaci. Jak je vidět z tabulky

Modul	Avg-t(s)	Max-t(s)	Min-t(s)
Překreslování	0.007	0.085	0.003
Zvuk (komplet)	0.069	0.963	0.04
CQT	0.054	0.836	0.036
Normalizace	0.0007	0.002	0.0005
Rozpoznávání	0.002	0.006	0.001

Tabulka 4.1: Rychlost jednotlivých modulů aplikace, kde: *Překreslování* - kompletní čas vykreslování nové obrazovky (GUI loop), *Zvuk komplet* - čas pro zpracování zvuku (bez načítání mikrofону), *CQT* - čas potřebný pro transformaci zvukového signálu pomocí CQT, *Normalizace* - čas potřebný pro normalizaci výsledku CQT transformaci, *Rozpoznávání* - čas potřebný pro inference neuronové sítě.

průměrná doba od začátku poslouchání do klasifikaci audio vzorků je *Zvuk (komplet) + 20ms = 0.089* sekund což se dá považovat za skoro real-time. Taková mala rychlost moc neobtěžuje systém, kvůli tomu, že grafika se kreslí nezávislé od zpracování zvuku. V případě když modul zpracování zvuku nestihne předat rozpoznané noty do modulu GUI, GUI bude používat rozpoznané noty z minulého okamžiku. A tím cela aplikace dokáže dosáhnout v průměru 142 FPS¹. Taký z této tabulky plně, že CQT transformace zabírá příliš hodně času. Kdyby v budoucnu se podařilo zmenšit čas na převodu zvukového signálu do frekvenční reprezentaci, pak by bylo možné použít větší a přísnější neuronovou síť.

¹FPS - Snímková frekvence (z anglického frames per second)

4.2 Přesnost klasifikaci

Po trénování malá neuronová síť byla testovaná na datech které nebyli v datasetu. Různé metriky přesnosti jsou znázorněny v tabulce 4.2. Z tabulky plně že síť dělá relativně

Metrika	Hodnota
Loss	0.023
Accuracy	0.47
f1	0.81
precision	0.87
recall	0.76

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

málo chyb ($Loss=0.023$), ale také plně že síť neprodukuje jisté odpovědi na vstupní data ($f1=0.81$). Pravděpodobně je to dano příliš krátkou dobou trénování. Což ve výsledné aplikaci se dá vyřešit zmenšeným prahem jistoty, kdy nota se považuje za detekovanou.

4.3 Hodnocení GUI

Aplikace byla poskytnuta 6 lidem s různým úrovněmi dovednosti hry na klavíru. Subjektivní hodnocení se dá shrnout do následujícího 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.’

4.4 Omezení

Podle provedených zkoušek aplikace může (subjektivně) dobře fungovat pouze pokud šum v prostředí je méně než 38 dB, pokud šum 40 dB a větší aplikaci je dost těžko použít.

Kapitola 5

Budoucí práce

Samozřejmě představená aplikace chce několik vylepšení a to jsou:

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

Kapitola 6

Závěr

Cílem dané diplomové prací bylo vytvoření aplikaci pro podporu cvičení hry na klavíru. Návrh takové aplikace představuje netriviální úlohu, zejména kvůli následujícím dvěma problémům. Prvním problémem je přesný překlad zvukového záznamu uživatele do vnitřní reprezentace, na základe které pak bude uživatel trénován. Druhým problémem je pořízení zvuku z mikrofону a kontrola v reálném čase zda to, co uživatel hraje, odpovídá originální nahrávce, aby aplikace 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 ve druhém případě, při real-time kontrole uživatele, je doba zpracování jednoho zvukového fragmentu na prvním místě. Oba dva problémy byli redukovány na problém klasifikace přehrávaných not podle fragmentu zvukového záznamu.

Naše řešení je bylo založené na použití dvou různé velikých neuronových síti (každá pro jednotlivé části programu) s různými architekturami. První neuronová síť je zaměřena na přesnou transkripci zvukové nahrávky do vnitřní reprezentaci notového zápisu - note batches. Dana síť s malými změny byla převzata z již existující prací [6]. Druhá síť byla vytvořená vlastními síly a použita za účelem průběžné kontroly zvuku z mikrofону, během cvičícího procesu. Velka část dané práci se zabývala vytvoření vlastní real-time neuronové síte: výběru architektury, formátu vstupních dat, přípravou datasetu a trénování.

Za účelem ověření přípustnosti nově navrženého řešení byla provedena série experimentu. Aplikace byla testovaná jak z časového hlediska tak i z hlediska použitelnosti konečným uživatelem. Pomoci těchto testu byla ověřena vhodnost zvolených architektur neuronových síti. Taky testy ukázali ze za určitých podmínek(omezení) aplikace je celkem funkční.

Literatura

- [1] 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.
- [2] HLAVSA, Z. et al. *Pravidla českého pravopisu*. 2. vyd. Academia, 2009. ISBN 80-200-1327-X.
- [3] KNUTH, D. E. *The T_EXbook*. Addison-Wesley Publishing Company, 1996. ISBN 0-201-13447-0.