VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2020

Bc. René Semela

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

AUTOMATICKÉ TAGOVÁNÍ HUDEBNÍCH DĚL POMOCÍ METOD STROJOVÉHO UČENÍ
AUTOMATIC TAGGING OF MUSICAL COMPOSITIONS USING MACHINE LEARNING METHODS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. René Semela
AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Kiska
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Audio inženýrství**

Ústav telekomunikací

Student: Bc. René Semela ***ID:*** 185943

Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Automatické tagování hudebních děl pomocí metod strojového učení

POKYNY PRO VYPRACOVÁNÍ:

V rámci této práce budou shrnuty dosavadní poznatky v oblasti tzv. Music Information Retrieval. Konkrétně poznatky věnující se automatickému tagování hudebních děl. V rámci této práce budou analyzovány hudební nahrávky z hlediska barvy zvuku, rytmiky a dynamiky za účelem tagování hudebních děl (např. interpret, skladba, žánr, nástrojové složení apod.). Cílem diplomové práce bude vytvořit žánrově pestrou databázi a následně navrhnout systém automatizovaného tagování využívající strojové učení. V rámci diplomové práce bude také tento systém implementován a otestován (např. na doporučování hudebního obsahu dle preferencí uživatele).

DOPORUČENÁ LITERATURA:

[1] Music similarity and retrieval: an introduction to audio- and web-based strategies. New York, NY: Springer Berlin Heidelberg, 2016. ISBN 9783662497203.

[2] MÜLLER, M. Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications [online]. Springer International Publishing Switzerland, 2015, 483 s. ISBN 978-3-319-21945-5.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Tomáš Kiska

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Systémy pro automatické tagování hudebních děl jsou jednou z mnoha výzev pro obor strojového učení, a to zejména z hlediska komplexnosti celé této problematiky. Praktické uplatnění mohou tyto systémy nalézat zejména v obsahové analýze hudebních děl nebo při třídění obsahu hudebních knihoven. Tato práce se zabývá návrhem, trénováním, testováním a evaluací architektur umělých neuronových sítí pro automatické tagování hudebních děl. V úvodu je pozornost věnována položení ucelených teoretických základů pro tuto problematiku. V praktické části je pak navrženo 8 architektur neuronových sítí (4 plně konvoluční a 4 konvolučně-rekurentní). Tyto architektury jsou následně natrénovány za pomoci MagnaTagATune Dataset a mel spektrogramu a následně je provedeno jejich testování a evaluace. Nejlepších výsledků zde dosahuje čtyřvrstvá konvolučně-rekurentní neuronová síť (CRNN4) s hodnotou ROC-AUC $= 0{,}9046 \pm 0{,}0016$. Jako další krok praktické části je vytvořen kompletně nový Last.fm Dataset 2020, který je sestaven díky napojení na API služeb Last.fm a Spotify. Tento nový dataset čítá 100 tagů a 122877 skladeb. Nejúspěšnější architektury jsou na tomto novém datasetu natrénovány, otestovány a evaluovány, a je tak položena základní hranice hodnot ROC-AUC, kterých lze za pomoci tohoto datasetu dosáhnout. Nejlepších výsledků zde dosahuje šestivrstvá plně konvoluční neuronová síť (FCNN6) s hodnotou ROC-AUC $= 0{,}8590 \pm 0{,}0011$. Na závěr celé práce je vytvořena jednoduchá aplikace pro otestování jednotlivých architektur neuronových sítí na uživatelem vloženém zvukovém souboru. Práce se svými výsledky vyrovnává světovým pracím na stejné téma a přináší několik nových poznatků a inovací. Z hlediska inovací je zejména dosaženo podstatného snížení komplexnosti jednotlivých architektur neuronových sítí v porovnání se světovými pracemi při zachování podobných výsledků.

KLÍČOVÁ SLOVA

automatické tagování, hudba, klasifikace, konvolučně-rekurentní neuronová síť, konvoluční neuronová síť, Last.fm Dataset 2020, MagnaTagATune Dataset, mel spektrogram, neuronová síť, obsahová analýza, rekurentní neuronová síť, strojové učení, získávání hudební informace, zpětnovazební neuronová síť

ABSTRACT

One of the many challenges of machine learning are systems for automatic tagging of music, the complexity of this issue in particular. These systems can be practically used in the content analysis of music or the sorting of music libraries. This thesis deals with the design, training, testing, and evaluation of artificial neural network architectures for automatic tagging of music. In the beginning, attention is paid to the setting of the theoretical foundation of this field. In the practical part of this thesis, 8 architectures of neural networks are designed (4 fully convolutional and 4 convolutional recurrent). These architectures are then trained using the MagnaTagATune Dataset and mel spectrogram. After training, these architectures are tested and evaluated. The best results are achieved by the four-layer convolutional recurrent neural network (CRNN4) with the ROC-AUC $= 0.9046 \pm 0.0016$. As the next step of the practical part of this thesis, a completely new Last.fm Dataset 2020 is created. This dataset uses Last.fm and Spotify API for data acquisition and contains 100 tags and 122877 tracks. The most successful architectures are then trained, tested, and evaluated on this new dataset. The best results on this dataset are achieved by the six-layer fully convolutional neural network (FCNN6) with the ROC-AUC $= 0.8590 \pm 0.0011$. Finally, a simple application is introduced as a concluding point of this thesis. This application is designed for testing individual neural network architectures on a user-inserted audio file. Overall results of this thesis are similar to other papers on the same topic, but this thesis brings several new findings and innovations. In terms of innovations, a significant reduction in the complexity of individual neural network architectures is achieved while maintaining similar results.

KEYWORDS

auto-tagging, automatic tagging, autotagging, classification, content analysis, convolutional neural network, convolutional recurrent neural network, Last.fm Dataset 2020, machine learning, MagnaTagATune Dataset, mel spectrogram, music, music information retrieval, neural network, recurrent neural network

SEMELA, René. *Automatické tagování hudebních děl pomocí metod strojového učení*. Brno, 2020, 103 s. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Tomáš Kiska.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Automatické tagování hudebních děl pomocí metod strojového učení“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Tomášovi Kiskovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Výpočetní prostředky byly poskytnuty v rámci projektu e-Infrastruktura CZ (e-INFRA LM2018140) poskytovaného v rámci programu Velké infrastruktury pro výzkum, vývoj a inovace.

Další výpočetní prostředky byly poskytnuty Ústavem telekomunikací při Fakultě elektrotechniky a komunikačních technologií Vysokého učení technického v Brně.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Problematika *automatického tagování hudebních děl* se stala v posledních letech jednou z mnoha vyhledávaných výzev pro výzkumné týmy z celého světa, a to zejména díky masivnímu rozvoji *strojového učení*, které je silně spjato se stále zvyšujícím se výpočetním výkonem. V tomto ohledu je třeba zmínit zejména masivní rozmach *umělých neuronových sítí* a neustálé zvyšování jejich komplexnosti.

Samotné automatické tagování hudebních děl je velmi podstatnou součástí oborů zabývajících se obsahovou analýzou hudebních nahrávek a do jisté míry úzce souvisí s automatickou klasifikací žánrů, nálad a podobně. Pro klasifikaci žánrů a nálad, kdy je úloha klasifikace silně soustředěna jedním konkrétním směrem, je velice efektivní využití metod tradičního strojového učení (např. GMM nebo SVM). Při úlohách automatického tagování, kdy jsou kategorie jednotlivých tagů velmi různorodé (*žánr*, *nálada*, *nástrojové obsazení*, *hlasitost* apod.), však tradiční strojové učení naráží na jistá omezení. S příchodem umělých neuronových sítí však tato omezení začala ztrácet na významu, a byla tak otevřena cesta k řešení stále více komplexních úloh.

Tato práce se zabývá návrhem, trénováním, testováním a evaluací neuronových sítí sloužících jako *multi-label klasifikátor* pro automatické tagování hudebních děl. Pro trénování neuronových sítí jsou zde postupně použity dva datasety (*MagnaTagATune Dataset* – všeobecně nejpoužívanější dataset pro řešení této problematiky, *Last.fm Dataset 2020* – nově vytvořený dataset jako součást řešení této práce). Použitím historicky zavedeného datasetu je zajištěna zejména možnost porovnávat dosažené výsledky s výsledky dosaženými v jiných pracích zabývajících se stejnou nebo podobnou problematikou. Díky testování na kompletně novém datasetu jsou pak vyvozeny zcela nové závěry, které vyvstaly do popředí díky použití odlišných dat.

První kapitola stručně shrnuje problematiku oboru zvaného *získávání hudebních informací* (MIR) a jsou zde zároveň popsány algoritmy využívané pro parametrizaci hudebních signálů v průběhu řešení této práce.

Těžiště celé teoretické části leží v druhé kapitole, která je zpočátku věnována tradičnímu strojovému učení a jsou zde popsány *klasifikační algoritmy* jako *k-nejbližších sousedů* (k-NN), *Gaussovy smíšené modely* (GMM) nebo *metoda podpůrných vektorů* (SVM). Stěžejní část této kapitoly pak tvoří podrobný popis neuronových sítí z hlediska jejich *elementárních stavebních bloků*, *vrstev*, *architektur*, *ztrátových funkcí* a *optimalizačních algoritmů*. Závěr kapitoly je následně tvořen popisem metod využívaných při testování a evaluaci klasifikátorů a věnuje se zejména křivce ROC a hodnotě ROC-AUC.

Třetí kapitola je věnována přiblížení problematiky automatického tagování hudebních děl z hlediska metodologie a technik využívaných pro tuto konkrétní formu klasifikace.

Čtvrtá kapitola následně stručně shrnuje výsledky, které byly dosaženy při řešení stejné nebo podobné problematiky v citované literatuře.

Jádro celé této práce pak spočívá v páté kapitole, která je věnována použitým technologiím, předzpracování prvního z datasetů (*MagnaTagATune Dataset*) a návrhu osmi

architektur neuronových sítí včetně jejich trénování. Na závěr této kapitoly je celý proces trénování všech architektur stručně shrnut.

Šestá kapitola je věnována testování a evaluaci všech osmi architektur neuronových sítí a na závěr této kapitoly je pozornost věnována porovnání dosažených výsledků s výsledky dosaženými v citované literatuře ze čtvrté kapitoly.

Sedmá kapitola je pak věnována trénování, testování a evaluaci nejlepších architektur neuronových sítí z páté kapitoly na nově vytvořeném datasetu (*Last.fm Dataset 2020*). V závěru této kapitoly jsou pak odhaleny nové poznatky, které s použitím prvního datasetu odhaleny nebyly.

Osmá a zároveň závěrečná kapitola pak pouze stručně popisuje vytvořenou aplikaci pro otestování jednotlivých architektur neuronových sítí na uživatelem zvoleném zvukovém souboru.

1 Získávání hudebních informací

Text této kapitoly pojednává o teoretickém úvodu do oboru zvaného *získávání hudebních informací* (anglickým názvem *Music Information Retrieval*, zkráceně MIR), jehož základy byly položeny v 90. letech 20. století jako odpověď na stále vzrůstající všeobecnou potřebu analyzovat a klasifikovat hudební signály [1, 2].

V užším kontextu lze MIR charakterizovat jako jeden z podoborů tradiční analýzy signálů. Na vstup analyzujícího systému je přiveden vstupní signál ve formě posloupnosti zvukových vzorků a na výstupu tohoto systému jsou následně získány hodnoty určitého parametru, který je daným systémem definován [2].

Výše uvedený princip však popisuje pouze jeden z mnoha aspektů získávání hudebních informací, a tím je *extrakce parametrů*. V širším kontextu je potřeba MIR vnímat jako spojnici mezi mnoha různorodými obory, jako jsou např. *analýza signálů*, *hudební teorie*, *muzikologie*, *psychologie* nebo *statistika*. Předmětem analýzy zároveň nemusí být výhradně čistě hudební signály, ale například i textová data nebo MIDI reprezentace [1].

1.1 Parametrizace hudebních signálů

Jak již bylo řečeno v úvodu této kapitoly, jednou ze základních a nedílných součástí MIR je *extrakce parametrů* ze zvukových signálů. Tento proces je v praxi zaváděn zejména za účelem redukce objemu dat pro další zpracovávání. Data obsažená ve zvukových signálech jsou totiž ze své podstaty značně redundantní.

Pro demonstraci redundance zvukových dat uvažujme například zvukový signál o vzorkovací frekvenci 44,1 kHz (CD standard) a podvzorkujme ho dvakrát na vzorkovací frekvenci 22,05 kHz, čímž je objem dat snížen na polovinu. Při běžném poslechu bude zcela jistě znatelný pokles kvality nahrávky z důvodu snížení maximální přenášené frekvence signálu podle vzorkovacího teorému. Informace obsažené v dané nahrávce (*text*, *nástrojové obsazení* apod.) však bude možné nadále rozeznat bez větších obtíží i přes výše zmíněnou redukci dat.

Základní dělení parametrů lze realizovat mnoha různými způsoby. Jedním z nejfrekventovanějších je rozdělení podle úrovně [1]. *Nízkoúrovňové parametry* jsou zpravidla získávány přímo ze zvukových vzorků signálu. Velmi často také slouží jako elementární stavební bloky pro výpočet parametrů vyšších úrovní. *Středněúrovňové parametry* potom popisují informace vztažené k výšce tónů nebo k tempu. Jsou zde zároveň zařazeny i složitější algoritmy jako např. *mel-frekvenční kepstrální koeficienty*, *počátky notových událostí* a další. *Vysokoúrovňové parametry* pak popisují nejkomplexnější informace o zkoumaném signálu. Do této kategorie lze zařadit např. *rozpoznávání melodie*, *rozpoznávání akordů*, *rozpoznávání rytmu*, *rozpoznávání nástrojového obsazení* a další [1].

Stručný přehled často používaných parametrů včetně jejich úrovně je pro názornost uveden v tab. 1.1. Podrobnější informace o konkrétních parametrech lze nalézt v literatuře [1, 2, 3, 4, 5]. Pro účely této práce jsou však informace obsažené v této kapitole zcela dostačující.

Název parametru	**Úroveň**
Amplitudová obálka	Nízká
Efektivní hodnota	Nízká
Počet průchodů nulovou úrovní	Nízká
Počátky notových událostí	Střední
Mel-frekvenční kepstrální koeficienty	Střední
Tempo	Střední
Rozpoznávání akordů	Vysoká
Rozpoznávání melodie	Vysoká
Rozpoznávání tóniny	Vysoká

Tab. 1.1: Výběr často používaných MIR parametrů a jejich úrovní.

1.1.1 Parametry využívané pro automatické tagování hudebních děl

Účelem této kapitoly je seznámit čtenáře s nejpoužívanějšími parametry využívanými při snaze o automatické tagování hudebních děl. Prvně je popsán a matematicky definován *spektrogram* a jeho modifikace ve formě *mel spektrogramu*. Právě spektrogram a jeho modifikace však není zcela vhodné označovat jako parametry – prakticky se totiž jedná pouze o *časově-frekvenční reprezentace* signálů. Na závěr jsou pak popsány *mel-frekvenční kepstrální koeficienty*.

Spektrogram

Spektrogram je jedním ze základních nástrojů pro analýzu zvukových signálů. Jedná se o reprezentaci daného signálu v časové i frekvenční oblasti [6].

Výpočet probíhá za pomoci algoritmu diskrétní STFT (*krátkodobá Fourierova transformace*, anglickým názvem *Short-Time Fourier Transform*). Na počátku je signál segmentován na krátké časové rámce o stejné délce a během tohoto procesu lze zvolit možnost překryvu jednotlivých rámců. V praxi se velmi často volí hodnota překryvu 50 %, což znamená, že nový rámec začíná vždy v polovině rámce předchozího. Následně jsou jednotlivé rámce násobeny symetrickým časovým oknem o stejné délce a na jednotlivé rámce je aplikován algoritmus DFT (*diskrétní Fourierova transformace*, anglickým názvem *Discrete Fourier Transform*), který je v praxi takřka výhradně realizován pomocí algoritmu FFT (*rychlá Fourierova transformace*, anglickým názvem *Fast Fourier Transform*) [6].

Diskrétní STFT lze definovat vztahem

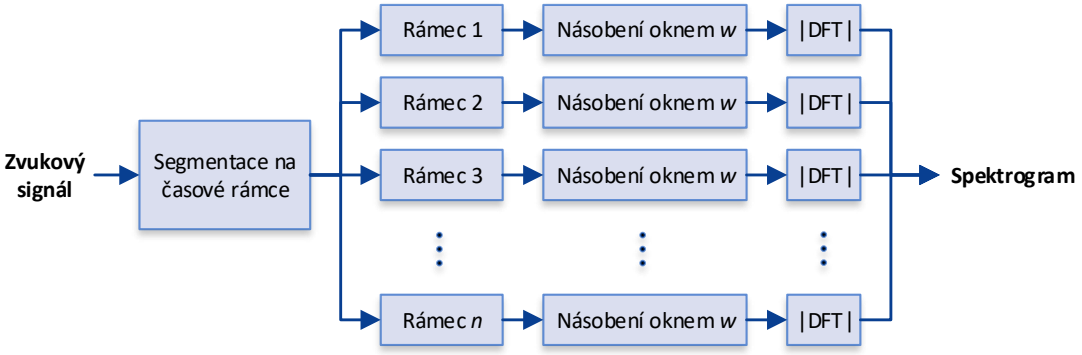
$$X_{\mathrm{STFT}}(m, f) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j2\pi fn}, \tag{1.1}$$

kde x je vstupní signál, w je časové okno (např. *Hann*, *Hamming* apod.), f je frekvence a m je posunutí [7]. Při výpočtu STFT se zároveň uplatňuje *Heisenbergův princip neurčitosti* modifikovaný D. Gaborem:

$$\Delta t \Delta f \geq \frac{1}{2}, \tag{1.2}$$

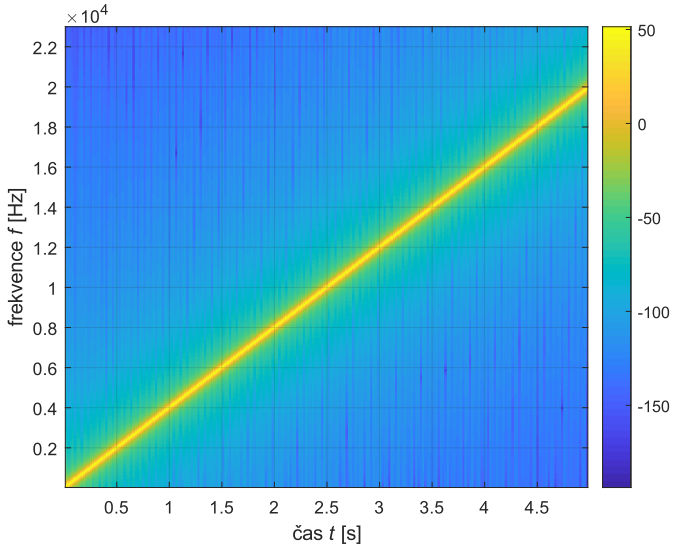
kde Δt je časové rozlišení a Δf frekvenční rozlišení [7, 8]. Ze vztahu (1.2) vyplývá, že se zvyšujícím se rozlišením v časové oblasti klesá rozlišení ve frekvenční oblasti a naopak [6].

Blokový diagram výpočtu spektrogramu je znázorněn na obr. 1.1. Parametry výpočtu spektrogramu jsou *délka časových rámců*, *překryv časových rámců*, *druh a délka okna* a *délka FFT*. V praxi jsou zpravidla všechny délky (*časové rámce*, *okno*, *FFT*) voleny jednotně.



Obr. 1.1: Blokové schéma výpočtu spektrogramu.

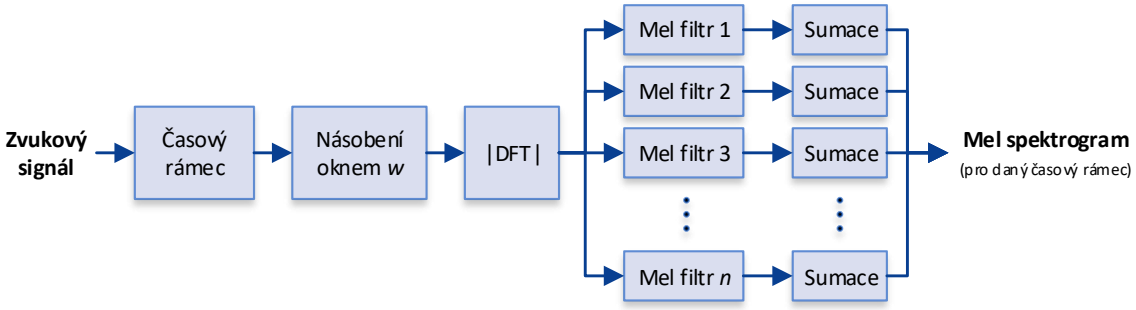
Na obr. 1.2 je pak znázorněna ukázka výstupu spektrogramu pro lineárně přeladovaný harmonický signál od 20 Hz do 20 kHz o délce trvání 5 s při vzorkovací frekvenci 48 kHz.



Obr. 1.2: Ukázka výstupu spektrogramu.

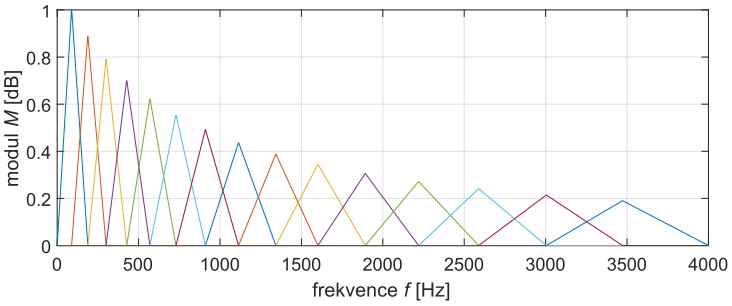
Mel spektrogram

Mel spektrogram je jednou z mnoha modifikací klasického spektrogramu popsaného v předchozí podkapitole. Časový rámec je po provedení STFT filtrován bankou *mel filtrů* a výsledné hodnoty po filtraci jsou v rámci jednotlivých filtrů sečteny. Aplikováním tohoto postupu na všechny časové rámce je získán mel spektrogram o rozměrech *počet mel filtrů* × *počet časových rámců* [9, 10]. Blokové schéma výpočtu mel spektrogramu pro jeden časový rámec je znázorněno na obr. 1.3 [10].

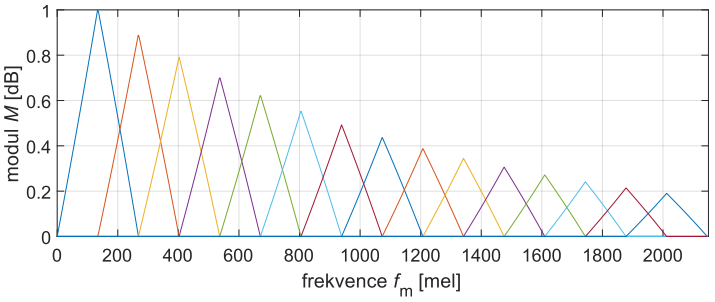


Obr. 1.3: Blokové schéma výpočtu mel spektrogramu.

Samotná banka filtrů je povětšinou realizována pomocí *trojúhelníkových filtrů*, které jsou považovány za nejjednodušší aproximaci *sluchových filtrů*. Mel škála je použita za účelem napodobení logaritmických vlastností lidského sluchu, kdy je rozložení vnímaných frekvencí na bazilární membráně rovněž logaritmické. Rozmístění filtrů je pak v mel škále pravidelné (viz obr. 1.4) [11].



(a) Lineární frekvenční osa.



(b) Mel frekvenční osa.

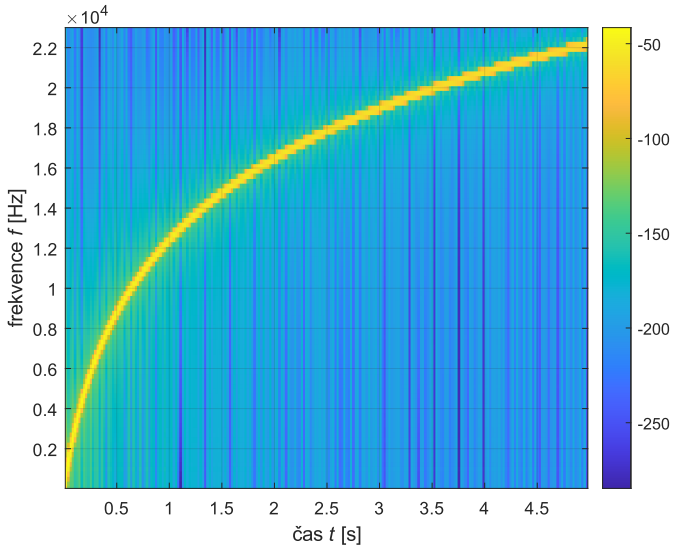
Obr. 1.4: Modulové frekvenční charakteristiky banky 15 trojúhelníkových mel filtrů.

Pro přepočet mezi frekvenční a mel škálou platí

$$f_{\mathrm{m}} = 2595 \log_{10}(1 + \frac{f}{700}), \tag{1.3}$$

kde f_{m} odpovídá frekvenci f v mel škále [2, 11].

Na obr. 1.5 lze názorně pozorovat, jak banka 96 mel filtrů ovlivnila grafický výstup mel spektrogramu pro lineárně přeladovaný harmonický signál.

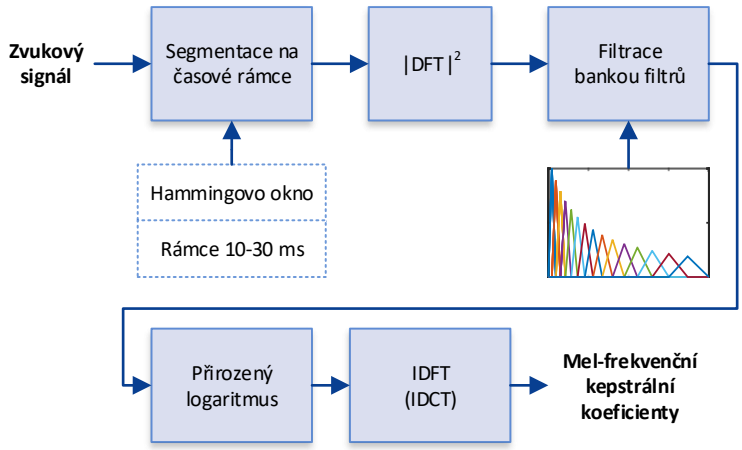


Obr. 1.5: Ukázka výstupu mel spektrogramu.

Mel-frekvenční kepstrální koeficienty

Mel-frekvenční kepstrální koeficienty (anglickým názvem *Mel-Frequency Cepstral Coefficients*, zkráceně MFCC) patří k velice populárním parametrům v oboru zpracování řeči i v oboru zpracování hudebních signálů. Stejně jako mel spektrogram je tento parametr založen na mel frekvenční škále a jsou zde využity stejné trojúhelníkové filtry (viz obr. 1.4).

Algoritmus pro výpočet MFCC je názorně zobrazen na obr. 1.6. Vstupní signál je segmentován na časové rámce a ty jsou následně transformovány pomocí DFT (v podobě FFT) na výkonové spektrum. Toto spektrum je následně filtrováno bankou mel filtrů, výstupy jednotlivých filtrů jsou sečteny a pro každý filtr je získána jedna jediná hodnota. Následně je proveden přirozený logaritmus a inverzní DFT těchto hodnot. Provedení inverzní DFT je realizováno pomocí *inverzní diskrétní kosinové transformace* (IDCT). Výstupem jsou pak *mel-frekvenční kepstrální koeficienty* [2, 11].



Obr. 1.6: Blokové schéma výpočtu mel-frekvenčních kepstrálních koeficientů.

2 Strojové učení

Strojové učení je jednou z mnoha podoblastí *umělé inteligence*. Jedná se o schopnost počítačového systému učit se a nabývat zkušenosti pomocí změn svých vnitřních stavů na základě poskytnutých podnětů.

Základní a typické dělení strojového učení lze realizovat následovně:

- *učení s učitelem* – během učení lze získat informace o chybách,
- *učení bez učitele* – během učení nelze získat informace o chybách (typicky *shlukování*) [11].

Z hlediska využití lze strojové učení dále rozdělit na úlohy týkající se *klasifikace* (zařazení vstupu do některé z výstupních tříd), *regrese* (odhad číselné hodnoty výstupu na základě vstupu) a *shlukování* (zařazování vstupů do skupin s podobnými charakteristikami). Pro účely této práce však bude následující obsah omezen pouze na *klasifikaci*, kterou lze dále rozdělit následovně:

- *klasifikace založená na vzdálenostech* – je vypočtena vzdálenost mezi známým a neznámým vzorem pomocí některé z *metrik* a na základě této vzdálenosti je neznámý vzor zařazen do některé ze tříd,
- *statistická klasifikace* – jsou využívány statistické modely, na jejichž základě je stanovena pravděpodobnost příslušnosti neznámého vstupního vzoru do konkrétních tříd [11].

V následujících textech se *příznakem* rozumí zejména data získaná pomocí parametrizace signálu (viz kap. 1.1). *Vzorem* se pak rozumí jeden konkrétní signál nebo data.

2.1 Klasifikační algoritmy

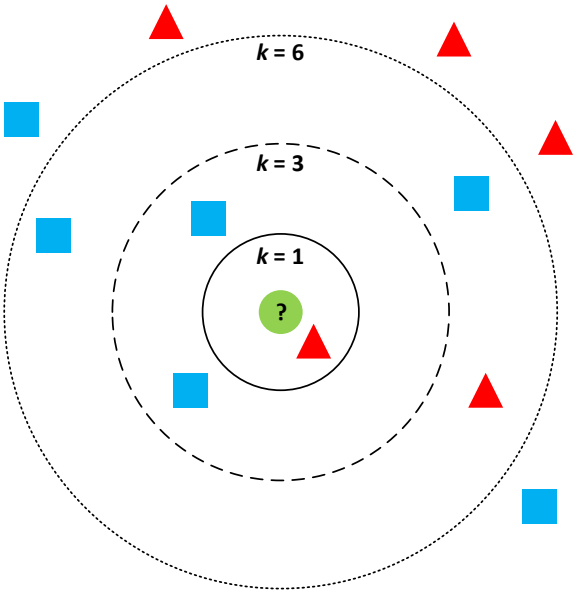
2.1.1 *k*-nejbližších sousedů

Algoritmus *k-nejbližších sousedů* (anglickým názvem *k-Nearest Neighbors*) je nejtypičtějším zástupcem ze skupiny vzdálenostních klasifikátorů. Pro neznámý vzor jsou získány vektory příznaků a jsou měřeny vzdálenosti vůči příznakům známých vzorů. K tomu je většinou využívána *Euklidovská vzdálenost* podle vztahu

$$D_{\mathrm{E}}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}, \tag{2.1}$$

kde $\boldsymbol{x} = (x_1, x_2, x_3, ..., x_n)$ je vektor příznaků pro neznámý vzor a $\boldsymbol{y} = (y_1, y_2, y_3, ..., y_n)$ je vektor příznaků pro známý vzor [2].

Na základě vzdáleností je následně neznámý vzor zařazen do třídy k nejbližších známých vzorů. Celý proces je názorně zobrazen na obr. 2.1. Pro $k = 1$ je neznámý vzor označen jako červený trojúhelník, pro $k = 3$ a $k = 6$ je pak zařazen jako modrý čtverec [2, 5].



Obr. 2.1: Určení výstupní třídy podle k nejbližších sousedů.

2.1.2 Gaussovy smíšené modely

Gaussovy smíšené modely (anglickým názvem *Gaussian Mixture Models*, zkráceně GMM) jsou jedním z nejčastěji využívaných statistických klasifikátorů. V principu se jedná o modelování množiny trénovacích příznaků jednou nebo více Gaussovými funkcemi. Výsledný smíšený model potom vznikne lineární kombinací m Gaussových funkcí a platí pro něj vztah

$$f_{\mathrm{GMM}}(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i f_i(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i \frac{1}{\sqrt{(2\pi)^n \det(\boldsymbol{K}_i)}} \exp\left(- \frac{(\boldsymbol{x}_i - \boldsymbol{\mu}_i)^{\mathrm{T}} \boldsymbol{K}_i^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_i)}{2} \right), \quad (2.2)$$

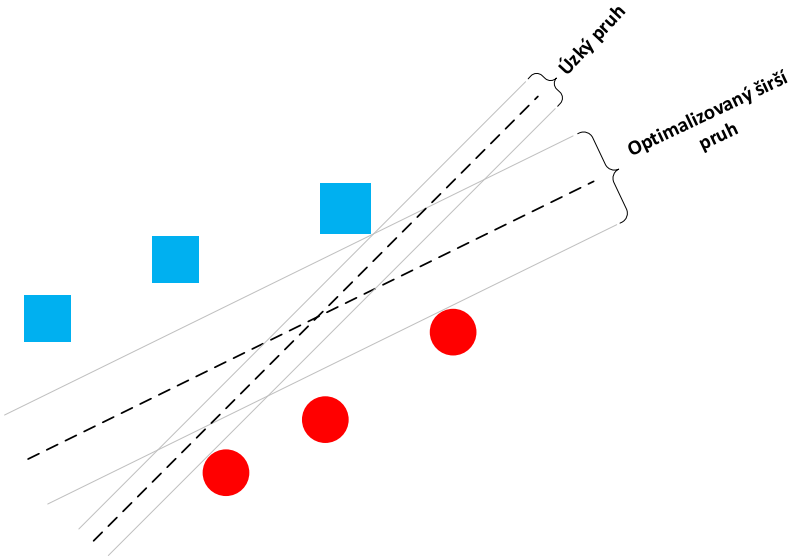
kde $\boldsymbol{x} = (x_1, x_2, x_3, ..., x_n)$ je vektor příznaků, $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3, ..., \mu_n)$ je vektor středních hodnot, n je počet příznaků, α je váhovací činitel, T znamená transpozici matice nebo vektoru a $\boldsymbol{K}$ je kovarianční matice, pro kterou platí

$$\boldsymbol{K} = \begin{bmatrix} E((x_1 - \mu_1)(x_1 - \mu_1)) & E((x_1 - \mu_1)(x_2 - \mu_2)) & \cdots & E((x_1 - \mu_1)(x_n - \mu_n)) \\ E((x_2 - \mu_2)(x_1 - \mu_1)) & E((x_2 - \mu_2)(x_2 - \mu_2)) & \cdots & E((x_2 - \mu_2)(x_n - \mu_n)) \\ E((x_3 - \mu_3)(x_1 - \mu_1)) & E((x_3 - \mu_3)(x_2 - \mu_2)) & \dots & E((x_3 - \mu_3)(x_n - \mu_n)) \\ \vdots & \vdots & & \vdots \\ E((x_n - \mu_n)(x_1 - \mu_1)) & E((x_n - \mu_n)(x_2 - \mu_2)) & \dots & E((x_n - \mu_n)(x_n - \mu_n)) \end{bmatrix}, \quad (2.3)$$

kde E značí střední hodnotu [11].

2.1.3 Metoda podpůrných vektorů

Metoda podpůrných vektorů (anglickým názvem *Support Vector Machine*, zkráceně SVM) je jedním z dalších populárních statistických klasifikátorů. Učení probíhá s učitelem a v principu je hledána nadrovina, která optimálně rozděluje množinu příznaků v daném prostoru do dvou poloprostorů. Optimální nadrovina je taková, okolo které vznikne nejširší pruh bez jakýchkoli příznaků. Tento proces je názorně vyobrazen na obr. 2.2 [5].



Obr. 2.2: Ukázka optimálního rozdělení prostoru pomocí algoritmu SVM.

Jedná se o *binární klasifikátor* – to znamená, že jsou data rozdělována pouze do dvou tříd. V případě použití SVM pro klasifikaci dat do více výstupních tříd je potřeba tuto úlohu rozdělit na elementární binární klasifikace podle jednoho z následujících pravidel:

- 1 vs. 1 – jsou postupně porovnávány všechny dvojice tříd vůči sobě,
- 1 vs. zbytek – je porovnávána vždy jedna třída vůči zbytku.

Výsledek klasifikace je potom rozhodnut na základě výsledků dílčích klasifikátorů [5].

2.2 Umělé neuronové sítě

Umělé neuronové sítě (anglickým názvem *Artificial Neural Networks*, zkráceně ANN nebo pouze NN) jsou jednou z podoblastí *umělé inteligence* (resp. *strojového učení*). Historie tohoto odvětví sahá až do roku 1943, kdy byl zkonstruován první matematický model umělého neuronu, za jejímž vznikem stála dvojice amerických vědců – W. Pitts a W. McCulloch [12]. Tento model napodoboval chování reálných neuronů obsažených v lidských neuronových sítích. V roce 1949 byl pak D. Hebbem představen způsob, kterým bylo možné tento jednoduchý neuron učit [13]. První neuronová síť byla následně zkonstruována v roce 1962 F. Rosenblattem [14]. Největší rozmach tohoto odvětví je však datován až od 80. let 20. století a je silně spojen s rapidním rozvojem výpočetních technologií. Nutno

podotknout, že popularita neuronových sítí roste dále i v současnosti. Použité technologie se však stávají stále složitějšími a sofistikovanějšími [15, 16].

Následující texty jsou omezeny pouze na relevantní druhy neuronových sítí z hlediska zadání této práce. Pro klasifikační úlohy jsou nejčastěji využívány *vícevrstvé neuronové sítě* (anglickým názvem *Multi-Layer Neural Networks*, zkráceně MLNN), které mohou, ale nemusí mít *zpětnou vazbu*. Dále je využíván algoritmus učení se *zpětným šířením chyby* (anglickým názvem *Back-Propagation of Gradient*, zkráceně BPG).

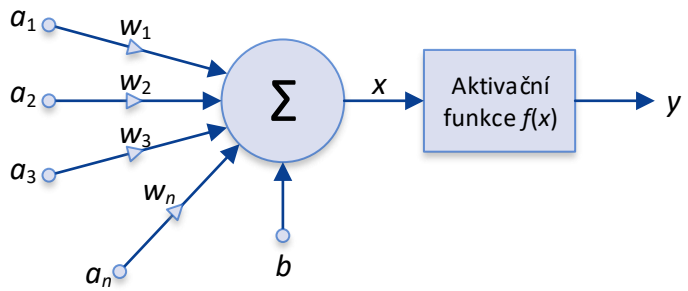
2.2.1 Model umělého neuronu

Model *umělého neuronu* vychází z biologického neuronu a je základní stavební jednotkou všech neuronových sítí. Neuron má zpravidla n vstupů, které jsou realizovány formou vektoru $\boldsymbol{a} = (a_1, a_2, a_3, ..., a_n)$. Každému vstupu pak přísluší váha z vektoru vah $\boldsymbol{w} = (w_1, w_2, w_3, ..., w_n)$. Tyto váhy ovlivňují míru příspěvku jednotlivých vstupů do vnitřního potenciálu neuronu, který je realizován sumační funkcí. Sumační funkce pak může být dále ovlivněna prahovou hodnotou b a výsledný vnitřní potenciál neuronu je následně určen podle vztahu

$$x = b + \sum_{i=1}^{n} a_i w_i. \tag{2.4}$$

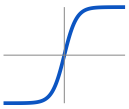
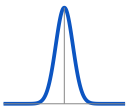
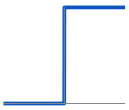
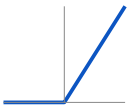
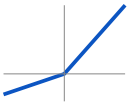
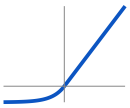
Výstup sumační funkce je zpravidla vstupem pro některou z nelineárních aktivačních funkcí $f(x)$, které určují způsob, kterým je vnitřní potenciál neuronu transformován. Z toho vyplývá, že konečný výstup neuronu y je závislý na druhu použité aktivační funkce [11, 15, 16].

Model umělého neuronu je graficky znázorněn na obr. 2.3 a přehled nejpoužívanějších aktivačních funkcí pak v tab. 2.1 [16].



Obr. 2.3: Model umělého neuronu.

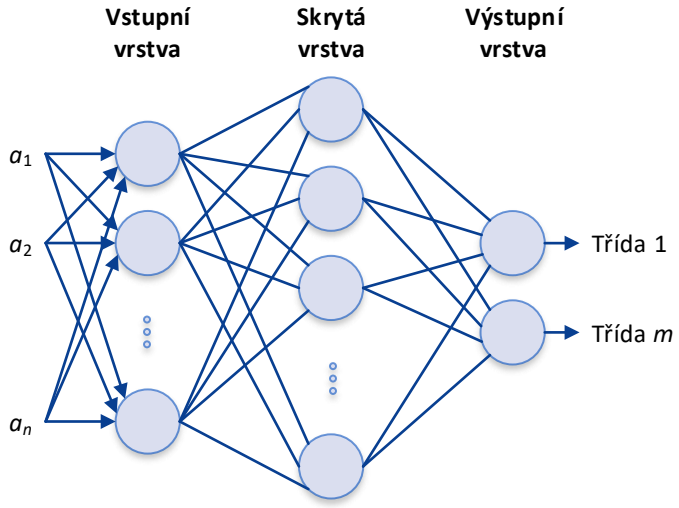
Tab. 2.1: Definice nejčastěji používaných aktivačních funkcí.

Název funkce	**Funkce**	**Tvar**
Sigmoida	$f(x) = \sigma(x) = \frac{1}{1+\mathrm{e}^{-x}}$	
Hyperbolický tangens	$f(x) = \tanh(x) = \frac{(\mathrm{e}^{x}-\mathrm{e}^{-x})}{(\mathrm{e}^{x}+\mathrm{e}^{-x})}$	
Gaussova funkce	$f(x) = \mathrm{e}^{-x^2}$	
Skoková funkce	$f(x) = \begin{cases} 0, & \text{pro } x < 0 \\ 1, & \text{pro } x \geq 0 \end{cases}$	
ReLU	$f(x) = \begin{cases} 0, & \text{pro } x < 0 \\ x, & \text{pro } x \geq 0 \end{cases}$	
PReLU	$f(x) = \begin{cases} \alpha x, & \text{pro } x < 0 \\ x, & \text{pro } x \geq 0 \end{cases}$	
ELU	$f(x) = \begin{cases} \alpha(\mathrm{e}^{x}-1), & \text{pro } x < 0 \\ x, & \text{pro } x \geq 0 \end{cases}$	

2.2.2 Vrstvy neuronových sítí

Jak již bylo řečeno v předchozí podkapitole, neuron jako takový je pouhou základní stavební jednotkou a nedokáže ze své podstaty řešit jakékoli složitější operace. Z toho důvodu byly zavedeny *vrstvy*, ve kterých jsou jednotlivé neurony uspořádány. Výstupy neuronů jedné vrstvy se pak mohou stát vstupy pro vrstvy následující, a tímto způsobem vznikají právě *neuronové sítě* [16].

Formálně se neuronové sítě skládají ze *vstupní vrstvy*, *skrytých vrstev* a *výstupní vrstvy* (viz obr. 2.4) [16].



Obr. 2.4: Ukázka jednoduché dopředné neuronové sítě.

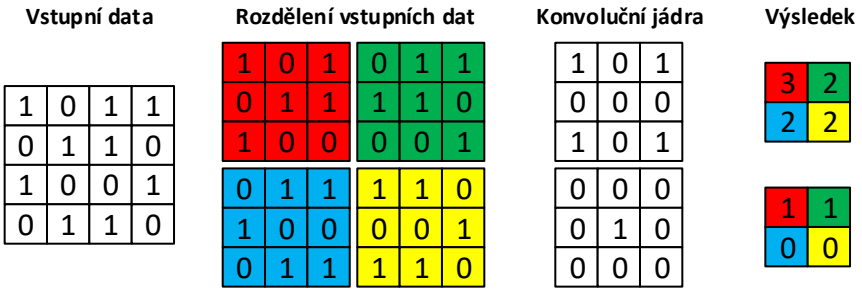
Jelikož v praxi existuje nemalé množství různých druhů vrstev, v této podkapitole jsou přiblíženy pouze ty, které přímo souvisejí se zadáním této práce. V úvodu zároveň nutno podotknout, že existuje i mnoho různých vrstev, které neobsahují žádné neurony a jejich úkolem je pouze specifická úloha jako např. *změna rozměru dat*, *podvzorkování dat* a podobně.

Plně propojená vrstva

Neurony této vrstvy jsou plně propojeny se všemi neurony vrstvy předchozí. Veškeré spoje jsou váhovány, čímž je opět řízen příspěvek jednotlivých vstupů do vnitřních potenciálů neuronů [16, 17].

Konvoluční vrstva

Na vstupní data je postupně aplikováno konvoluční *jádro/filtr* o zvolených rozměrech. Proces rozdělení vstupních dat a následnou aplikaci jader lze názorně pozorovat na obr. 2.5, kde je jádro posouváno po vstupních datech s nejmenším možným krokem (v praxi může být tento krok mnohonásobně větší). Po aplikování jádra jsou výsledné hodnoty sečteny, a tím je dán výsledek, jehož rozměr je úměrný použitému kroku a rozměru jádra [17, 18].



Obr. 2.5: Ukázka principu fungování konvoluční vrstvy.

Z matematického hlediska se jedná o prostou 2D konvoluci, kterou lze definovat podle vztahu

$$y[m,n] = x[m,n] * w[m,n] = \sum_k \sum_l x[k,l]w[m-k, n-l], \tag{2.5}$$

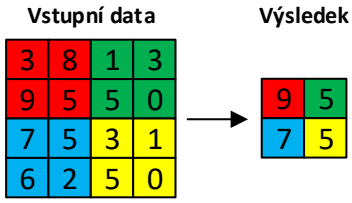
kde x jsou dvourozměrná vstupní data a w je dvourozměrné konvoluční jádro [7, 17].

V praxi je zpravidla použito několik jader najednou. V takovém případě je sice zmenšován rozměr výsledného pole, ale těchto polí vzniká díky použití jader více, a tím zároveň roste i komplexnost celé neuronové sítě. Příklad na obr. 2.5 znázorňuje použití dvou jader o rozměrech 3×3 na vstupní data o rozměrech 4×4 s krokem 1. Výsledek má v tomto případě rozměr $2 \times 2 \times 2$.

V případech, kdy nevyhovuje zvolený krok a zvolený rozměr jádra rozměrům vstupních dat a hrozilo by vynechání některých částí vstupních dat, je vhodné aplikovat tzv. *zero-padding*, který vstupní data na okrajích rozšíří o nulové hodnoty a zajistí tak, aby nebyly žádné informace obsažené ve vstupních datech vynechány.

Pooling vrstva

Úlohou vrstev typu *pooling* je redukce počtu hodnot. Z praktického hlediska se jedná o analogii k podvzorkování signálu. Podle toho, jakým způsobem dochází k redukci více hodnot do jedné, lze pooling vrstvy dělit na *max-pooling*, *avg-pooling*, *sum-pooling* a podobně. Princip funkce *max-pooling* vrstvy je názorně zobrazen na obr. 2.6 [17, 18].



Obr. 2.6: Ukázka principu fungování max-pooling vrstvy.

Batch Normalization vrstva

Batch Normalization vrstva vyhodnocuje pro dávku $\boldsymbol{b} = (x_1, x_2, x_3, ..., x_n)$ průměr a rozptyl. Následně jsou hodnoty této dávky normalizovány podle vzorce

$$\hat{x}_i = \frac{x_i - \mu_{\boldsymbol{b}}}{\sqrt{\sigma_{\boldsymbol{b}}^2 + \epsilon}}, \tag{2.6}$$

kde $\mu_{\boldsymbol{b}}$ odpovídá průměru a $\sigma_{\boldsymbol{b}}^2$ odpovídá rozptylu hodnot dávky $\boldsymbol{b}$ a ϵ je konstantou zajišťující numerickou stabilitu [21]. Dále jsou hodnoty posunuty a škálovány podle vzorce

$$y_i = \gamma \hat{x}_i + \beta, \tag{2.7}$$

kde γ a β jsou nové trénovatelné parametry neuronové sítě, které tato vrstva zavádí [17, 18, 21]. Tento proces ze své podstaty přispívá k redukci citlivosti na počáteční inicializaci parametrů neuronové sítě a zároveň přispívá ke zrychlení procesu trénování [21].

Dropout vrstva

Jedinou úlohou *dropout* vrstev je náhodná deaktivace určitého zvoleného procenta spojů mezi jednotlivými vrstvami, čímž je zajištěno zmenšení vzájemné závislosti jednotlivých neuronů. V praxi jsou tyto vrstvy zařazovány za účelem obrany proti *přeučení sítě* (anglickým názvem *overfitting*), kdy dochází k přílišné závislosti nastavení sítě na trénovací množině dat – pro neznámá data potom síť nedosahuje adekvátních výsledků [16, 17, 18].

Softmax vrstva

Softmax vrstva je zpravidla poslední vrstvou neuronových sítí a jejím účelem je transformace výstupů předchozí vrstvy na hodnoty pravděpodobností ležící v intervalu $\langle 0, 1 \rangle$. V klasifikačních úlohách odpovídá počet těchto hodnot počtu predikovaných tříd a jejich součet je roven 1. Tím se softmax vrstva stává ideální pro použití v tzv. *multi-class* klasifikaci, kdy vstupnímu vzoru náleží právě jedna třída [16, 17].

V případě *multi-label* klasifikace, kdy může jednomu vstupnímu vzoru odpovídat vícero tříd, není tedy její použití žádoucí, protože je zde kladen požadavek na nezávislost predikovaných pravděpodobností pro jednotlivé třídy.

2.2.3 Architektury neuronových sítí

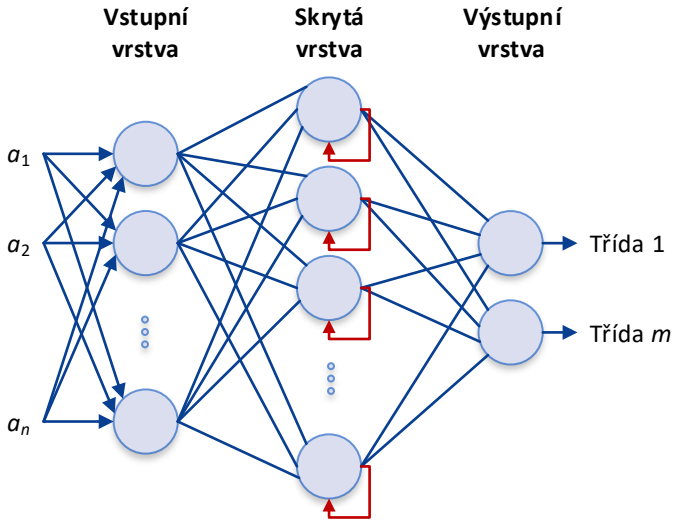
Neuronové sítě lze z hlediska jejich architektur dělit na *dopředné sítě*, *zpětnovazební sítě* a *samoorganizující sítě*. V následujících textech budou blíže přiblíženy především sítě zpětnovazební.

Princip dopředných vícevrstvých sítí byl již názorně popsán na obr. 2.4. Použitý počet neuronů i počet skrytých vrstev se vždy odvíjí od charakteru řešeného problému a od kvality a množství trénovacích dat.

Zpětnovazební neuronové sítě

Zpětnovazební neuronové sítě (anglickým názvem *Recurrent Neural Networks*, zkráceně RNN), jak již sám název napovídá, obsahují zpětné vazby, kterými jsou přiváděny výstupy neuronů zpět na jejich vlastní vstupy nebo na vstupy neuronů některé z předchozích vrstev. Hlavním důvodem pro zavádění těchto sítí je jejich schopnost dynamického zpracování informací [16]. V praxi se jedná zejména o fakt, že síť má lepší předpoklady pro nacházení souvislostí v rámci dlouhých úseků časově proměnných vstupních dat. Tím se tato architektura stává vhodnou volbou pro zpracování hudebních signálů.

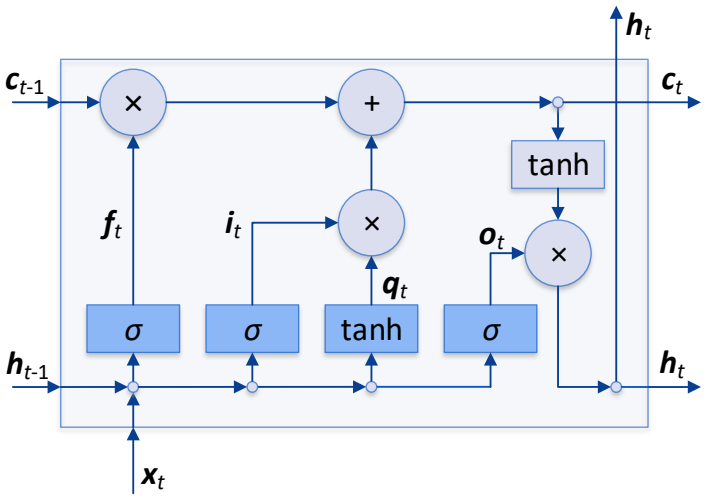
Na obr. 2.7 je znázorněn možný princip realizace zpětných vazeb [16].



Obr. 2.7: Ukázka realizace zpětných vazeb ve zpětnovazebních neuronových sítích.

2.2.4 LSTM sítě

LSTM sítě (zkratka slov *Long Short-Term Memory*) jsou jednou z konkrétních realizací zpětnovazebních neuronových sítí a jsou vhodné pro analýzu dlouhodobých časových závislostí. Jejich základním stavebním prvkem jsou LSTM *buňky*, které jsou následně řetězeny za sebe. Následující texty jsou čerpány z původní práce [22] a dále pak z [16, 23].



Obr. 2.8: Buňka LSTM sítě.

Nejdůležitější součástí buňky je její *vnitřní stav* $\boldsymbol{c}_t$, který je ovlivňován pouze lineárními operacemi za pomoci tří *bran*, které jsou realizovány neurony. Buňka na počátku dědí svůj vnitřní stav od buňky předchozí a tento stav je následně aktualizován.

Zapomínací brána (anglickým názvem *forget gate*) rozhoduje o tom, které informace budou odstraněny z vnitřního stavu buňky podle vztahu

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_\mathrm{f}\boldsymbol{x}_t + \boldsymbol{U}_\mathrm{f}\boldsymbol{h}_{t-1} + \boldsymbol{b}_\mathrm{f}), \tag{2.8}$$

kde $\boldsymbol{x}_t$ je vektor vstupních dat, $\boldsymbol{h}_{t-1}$ je vektor výstupu předchozí LSTM buňky, $\boldsymbol{W}_\mathrm{f}$ a $\boldsymbol{U}_\mathrm{f}$ jsou matice vah zapomínací brány a $\boldsymbol{b}_\mathrm{f}$ je vektor prahových hodnot zapomínací brány. Výstupem je hodnota v intervalu $\langle 0, 1\rangle$, kterou je řízena velikost vnitřního stavu buňky.

Vstupní brána (anglickým názvem *input gate*) pak řídí, jakými informacemi bude vnitřní stav buňky aktualizován podle vzorců

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_\mathrm{i}\boldsymbol{x}_t + \boldsymbol{U}_\mathrm{i}\boldsymbol{h}_{t-1} + \boldsymbol{b}_\mathrm{i}), \tag{2.9}$$

$$\boldsymbol{q}_t = \tanh(\boldsymbol{W}_\mathrm{q}\boldsymbol{x}_t + \boldsymbol{U}_\mathrm{q}\boldsymbol{h}_{t-1} + \boldsymbol{b}_\mathrm{q}). \tag{2.10}$$

Prvky použité ve ve vztazích (2.9) a (2.10) jsou analogické k prvkům použitým při výpočtu výstupu zapomínací brány.

V tuto chvíli lze již získat *nový vnitřní stav* buňky podle vztahu

$$\boldsymbol{c}_t = \boldsymbol{f}_t \times \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \times \boldsymbol{q}_t. \tag{2.11}$$

Výstupní brána je následně určena vztahem

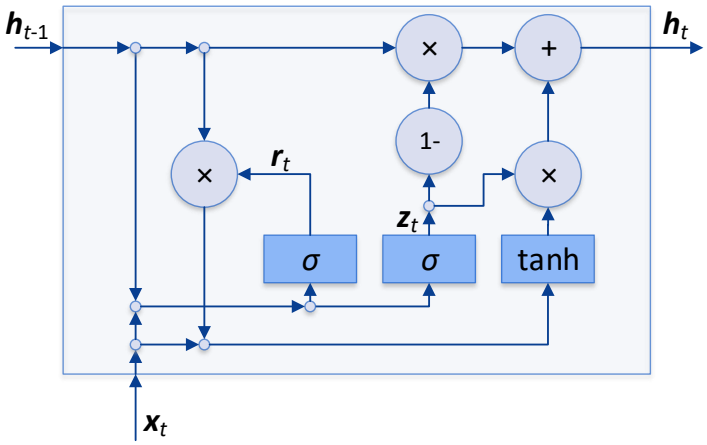
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_\mathrm{o}\boldsymbol{x}_t + \boldsymbol{U}_\mathrm{o}\boldsymbol{h}_{t-1} + \boldsymbol{b}_\mathrm{o}). \tag{2.12}$$

Celkový *výstup* LSTM buňky je pak získán z vnitřního stavu buňky a výstupní brány pomocí vztahu

$$\boldsymbol{h}_t = \tanh(\boldsymbol{c}_t) \times \boldsymbol{o}_t. \tag{2.13}$$

2.2.5 GRU sítě

GRU sítě (zkratka slov *Gated Recurrent Unit*) do značné míry vycházejí z architektury LSTM sítí. Jedna buňka sítě obsahuje dvě *brány*, které ovlivňují vnitřní stav lineárními operacemi. V případě GRU je vnitřní stav buňky na rozdíl od LSTM roven výstupu buňky. Jde o relativně nový princip, který byl publikován v roce 2014 v práci [24], ze které jsou následující texty čerpány.



Obr. 2.9: Buňka GRU sítě.

Zapomínací brána a vstupní brána jsou zde zkombinovány v jednu jedinou *aktualizační bránu* (anglickým názvem *update gate*), která odpovídá vztahu

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_\mathrm{z}\boldsymbol{x}_t + \boldsymbol{U}_\mathrm{z}\boldsymbol{h}_{t-1} + \boldsymbol{b}_\mathrm{z}), \tag{2.14}$$

kde $\boldsymbol{x}_t$ je vektor vstupních dat, $\boldsymbol{h}_{t-1}$ je vektor výstupu předchozí GRU buňky, $\boldsymbol{W}_\mathrm{z}$ a $\boldsymbol{U}_\mathrm{z}$ jsou matice vah aktualizační brány a $\boldsymbol{b}_\mathrm{z}$ je vektor prahových hodnot aktualizační brány.

Následně je získána odezva *resetovací brány* (anglickým názvem *reset gate*) podle vztahu

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_\mathrm{r}\boldsymbol{x}_t + \boldsymbol{U}_\mathrm{r}\boldsymbol{h}_{t-1} + \boldsymbol{b}_\mathrm{r}) \tag{2.15}$$

a výstup celé GRU buňky odpovídá vztahu

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \times \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \times \tanh(\boldsymbol{W}_\mathrm{h}\boldsymbol{x}_t + \boldsymbol{U}_\mathrm{h}(\boldsymbol{r}_t \times \boldsymbol{h}_{t-1}) + \boldsymbol{b}_\mathrm{h}). \tag{2.16}$$

2.2.6 Zpětné šíření chyby

Zpětné šíření chyby (anglickým názvem *Back-Propagation of Gradient*, zkráceně BPG) je jedním z nejčastěji používaných algoritmů pro trénování neuronových sítí. Tato metoda je založena na výpočtu chybovosti sítě a následné úpravě vah mezi jednotlivými neurony tak, aby byla chybovost sítě co nejmenší. Celý proces trénování neuronové sítě pomocí zpětného šíření chyby lze zjednodušeně popsat následovně:

1. inicializace vah sítě,
2. výběr vzoru z trénovací množiny,
3. určení odezvy sítě na vybraný vstupní vzor,
4. určení chyby na výstupu sítě,
5. určení chyby v předchozích vrstvách pomocí zpětného šíření chyby,
6. lokální úprava vah,
7. opakování celého procesu od bodu 2, dokud nejsou vlastnosti sítě přijatelné [19].

Matematicky se jedná o relativně složitý proces a je podrobněji popsán zejména v původní práci [20] a dále pak v [16, 17, 18].

2.2.7 Ztrátové funkce a optimalizační algoritmy

V předchozích podkapitolách byly popsány základní stavební prvky neuronových sítí, na základě kterých lze navrhnout a zkonstruovat neuronovou síť. Návrh sám o sobě však k úspěšnému natrénování sítě nestačí. K tomu slouží právě optimalizační algoritmy (anglickým názvem *optimizers*), které určují, jakým způsobem budou vnitřní stavy neuronové sítě aktualizovány na základě výsledků ztrátové funkce (anglickým názvem *loss function*). To vše za pomoci algoritmu *zpětného šíření chyby*.

Tato práce je omezena pouze na ztrátovou funkci *binární vzájemné entropie* (anglickým názvem *binary cross-entropy*), která je jednou z nejpoužívanějších ztrátových funkcí pro úlohy binární klasifikace a lze ji definovat pomocí vzorce

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}\Big(y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\Big), \tag{2.17}$$

kde n je počet vstupních vzorů, y_i je správné binární zařazení konkrétního vstupního vzoru a $\hat{y}_i$ je predikovaná pravděpodobnost zařazení vstupního vzoru [25].

Z hlediska optimalizačních algoritmů je v této práci využíván pouze algoritmus *Adam*, který byl poprvé popsán v práci [26] z roku 2014 a velmi rychle se stal standardem na poli optimalizačních algoritmů zejména díky faktu, že jsou v něm spojeny kladné vlastnosti jiných dříve hojně využívaných algoritmů jako např. *RMSprop* nebo *AdaGrad*. Z uživatelského hlediska je nejdůležitějším parametrem Adam algoritmu tzv. *learning rate*, jehož výchozí hodnota je nastavena na 0,001. Nastavení této hodnoty je kritickým faktorem, který ovlivňuje správné natrénování neuronové sítě. Matematický aparát na pozadí tohoto algoritmu je poměrně složitý a více informací lze najít v původní práci [26].

2.3 Evaluace klasifikátorů

Vyhodnocení úspěšnosti klasifikátoru je stěžejní informací o tom, s jakou spolehlivostí klasifikátor pracuje při zařazování neznámých dat. Evaluace je zpravidla prováděna na testovací množině dat, pro kterou jsou správné výsledky již známy. Tyto výsledky jsou následně porovnávány s výsledky klasifikátoru.

2.3.1 Přesnost

Za jeden ze základních evaluačních nástrojů lze považovat *přesnost* (anglickým názvem *Accuracy*, zkráceně ACC), kterou lze vyjádřit vztahem

$$ACC = \frac{\text{počet správných predikcí}}{\text{celkový počet predikcí}}. \tag{2.18}$$

Uvažujme tedy množinu testovacích dat, která obsahuje 100 vzorů. Tyto vzory jsou přivedeny na vstup klasifikátoru a do správných tříd je zařazeno 90 z nich. Výsledná přesnost tohoto klasifikátoru je pak podle vztahu (2.18) rovna 90 %.

2.3.2 ROC křivka

ROC křivka (zkratka slov *Receive Operating Characteristic*) je jedním z nejpoužívanějších prostředků ke zhodnocení kvality klasifikátorů. Před samotnou definicí je však vhodné zavést pomocné hodnoty podle tab. 2.2 [27].

Tab. 2.2: Pomocné hodnoty pro definici ROC křivky.

Zkratka	**Název**	**Popis**
TP	*True Positive*	Vstupní vzor je přiřazen třídě, do které patří.
TN	*True Negative*	Vstupní vzor není přiřazen třídě, do které nepatří.
FP	*False Positive*	Vstupní vzor je přiřazen třídě, do které nepatří.
FN	*False Negative*	Vstupní vzor není přiřazen třídě, do které patří.

V praxi se však pro definici hodnot z tab. 2.2 často využívá tzv. *matice záměn* (anglickým názvem *Confusion Matrix*), která je znázorněna tabulkou 2.3.

Tab. 2.3: Matice záměn.

	Skutečná 1	Skutečná 0
Predikovaná 1	*TP*	*FP*
Predikovaná 0	*FN*	*TN*

Hodnota *True Positive Rate* (zkráceně TPR) pak definuje *míru pravdivé pozitivity* a lze ji definovat podle vztahu

$$TPR = \frac{TP}{TP + FN} \quad [28]. \tag{2.19}$$

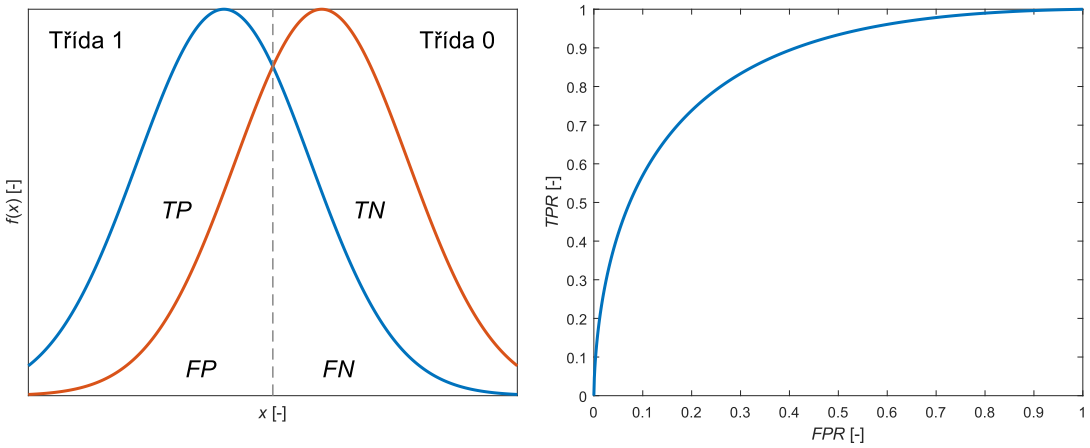
TPR je velmi často označováno též jako *senzitivita*.

Dále je vhodné definovat rovněž hodnotu *False Positive Rate* (zkráceně FPR), která určuje *míru falešné pozitivity* podle vztahu

$$FPR = \frac{FP}{FP + TN} \quad [28]. \tag{2.20}$$

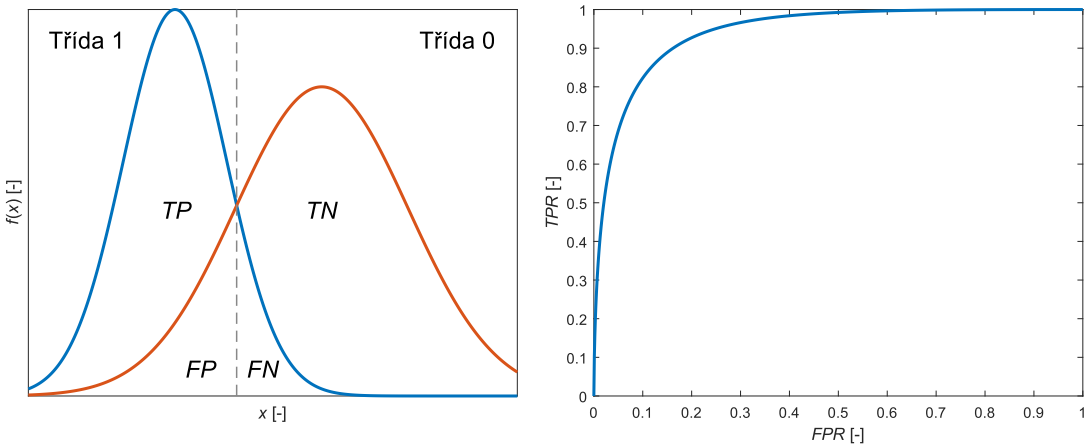
Pro úplnost je dále stěžejní zavést tzv. *klasifikační práh* (anglickým názvem *classification threshold*), který pro binární klasifikátor udává hodnotu, na základě které je rozhodnuto, jestli je daný vstupní vzor do konkrétní třídy zařazen, nebo ne [28].

Uvažujme binární klasifikaci testovací množiny, jejíž výsledky lze pro každou třídu vyjádřit pomocí hustoty pravděpodobnosti. ROC křivka je následně definována jako závislost TPR na FPR pro různé klasifikační prahy [28]. Pokud je rozlišitelnost hustot pravděpodobností na dobré úrovni, ROC křivka se v určitém bodě více přibližuje k bodu o souřadnicích $[0, 1]$. Tento bod udává optimální klasifikační práh a čím blíže se křivka k tomuto bodu přibližuje, tím je klasifikátor spolehlivější (viz obr. 2.10 a obr. 2.11) [27].



(a) Hustoty pravděpodobností. (b) ROC křivka.

Obr. 2.10: Ukázka špatné rozlišitelnosti tříd binárního klasifikátoru.



(a) Hustoty pravděpodobností. (b) ROC křivka.

Obr. 2.11: Ukázka dobré rozlišitelnosti tříd binárního klasifikátoru.

ROC-AUC

ROC-AUC (zkratka ze slov *Area Under the ROC Curve*) je v podstatě integrálem ROC křivky a udává spolehlivost klasifikátoru napříč všemi uvažovanými klasifikačními prahy.

$$ROC\text{-}AUC = \int_0^1 ROC(p)\mathrm{d}p \quad [27]. \tag{2.21}$$

Výhodou je, že výsledkem je pouhá jedna jediná hodnota, a tudíž je ROC-AUC vhodným nástrojem pro snadnou interpretaci výsledné kvality navrženého klasifikátoru.

3 Automatické tagování hudebních děl

Procesem *tagování* (anglickým názvem *tagging*, česky také *štítkování*) hudebních děl se rozumí přidělování krátkých textových řetězců, které dané skladby konkrétním způsobem charakterizují. Tyto tagy nalézají uplatnění zejména při třídění hudebních knihoven nebo při analýze hudebního obsahu skladeb.

Za popularizací tagování hudebních děl stojí zejména sociální sítě orientované na hudební obsah. Ukázkovým příkladem takové sítě může být například služba *Last.fm*[1], ve které dochází k tzv. *kolaborativnímu tagování*, během něhož mají uživatelé možnost přiřazovat tagy skladbám v tamní databázi. Na jejich základě je potom možné vyhledávat skladby nebo třídit hudební knihovny. V systémech založených na uživatelské komunitě však dochází k nedostatečnému pokrytí méně populárních skladeb a tento fakt lze považovat za jeden z hlavních důvodů ke snaze zavést systémy pro *automatické tagování* [1].

Obsah tagů je často velmi různorodý, a proto je lze zařazovat do mnoha různých kategorií, jako jsou například *žánr*, *nálada*, *nástrojové obsazení*, *tempo* a další. Při snaze o automatické přidělování tagů lze tedy narazit na souvislosti např. s automatickou klasifikací žánrů nebo nálady. Společným předmětem zájmu všech těchto odvětví je zcela jistě využití *strojového učení* [1]. Nicméně právě z důvodu značné různorodosti tagů se nabízí využívat mírně odlišné přístupy, než by tomu bylo při výše zmíněných klasifikacích.

3.1 Techniky automatického tagování

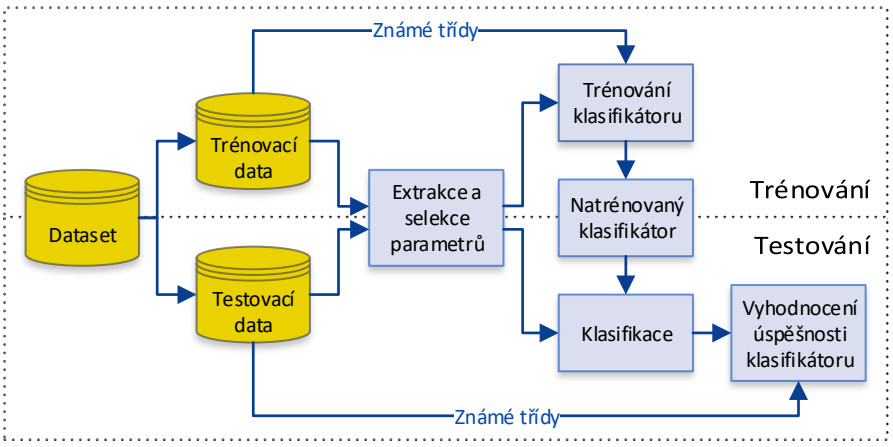
Systémy pro automatické tagování hudebních děl jsou výhradně realizovány za pomoci *klasifikátorů*. Jedná se o druh strojového učení, které zařazuje *vstupní vzor* do jedné nebo více *výstupních tříd*. V tomto případě mohou být vstupními vzory některé parametry z podkapitoly 1.1 – při takovém použití se tyto parametry často označují jako *příznaky* (anglickým názvem *features*). Výstupními třídami pak mohou být jednotlivé tagy nebo vektor pravděpodobností, s jakými vstupní vzor odpovídá jednotlivým tagům. Tento druh klasifikace je též označován jako *multi-label klasifikace* a jedná se v podstatě o kombinaci několika binárních klasifikátorů do jednoho celku (každému tagu náleží jeden binární klasifikátor).

Ke konstrukci těchto klasifikátorů jsou využívány tzv. *datasety*. Dataset je soubor již otagovaných dat, která jsou následně rozdělena na *trénovací* a *testovací* množinu. Trénovací množina je využita pro samotný proces *trénování* klasifikátoru. Testovací množina potom slouží ke zhodnocení kvality výsledného klasifikátoru. Testovací data jsou po natrénování klasifikátoru přivedena na jeho vstup jako data neznámá a výsledek klasifikace je porovnáván s již známými správnými třídami. Mimo trénovací a testovací množinu je velmi často využíváno rovněž i *validační* množiny. Jedná se o obdobu testovací množiny, která je však využita již v průběhu trénování pro validaci celého trénovacího procesu. V praxi se testovací a validační množiny velmi často zaměňují v jednu jedinou množinu, ale v rámci této práce bude vždy využito trénovacích, validačních i testovacích množin.

[1] `https://www.last.fm/`

U většiny reálných datasetů jsou výsledky testování často zavádějící, jelikož mohou být testovacím datům přiřazeny i tagy, které jsou z praktického hlediska sice správné, nicméně z formálního hlediska tyto tagy nemusí být skladbám v datasetu přiřazeny, a tudíž jsou výsledky vyhodnoceny jako chybné [1]. Pro názornost uvažujme na vstupu klasifikátoru pomalou barokní kytarovou skladbu, která je součástí testovacích dat a podle datasetu jí náleží 2 tagy – *kytara* a *baroko*. Na výstupu klasifikátoru se však objeví 3 tagy – *kytara*, *baroko* a *pomalé*. Tag *pomalé* je v tomto případě vyhodnocen jako špatné zařazení, nicméně z praktického hlediska je vše správně. Tento jev se následně negativně projeví na konečném hodnocení robustnosti daného klasifikátoru. Je tedy žádoucí tento jev co nejvíce eliminovat zvolením kvalitního datasetu nebo kvalitním předzpracováním jeho dat.

Základní princip trénování a testování klasifikátoru je znázorněn na obr. 3.1 [1, 11]. Za pozornost stojí zejména blok *exktrakce a selekce parametrů*. V praxi probíhá výběr vstupních parametrů pro klasifikátor tak, že pro trénovací data je vypočteno velké množství parametrů, ze kterých jsou následně vybrány ty nejrelevantnější pro danou aplikaci. Samotná selekce parametrů je pak často realizována algoritmem mRMR (zkratka slov *minimum Redundancy Maximum Relevance*) nebo SF(B)FS (zkratka slov *Sequential Forward (Backward) Floating Search*) [2, 5, 11]. Při použití neuronových sítí je však v praxi často využíváno velmi nízkoúrovňových parametrů. Z tohoto důvodu pak odpadá potřeba selekce parametrů a je zpravidla pracováno pouze s jedním konkrétním parametrem.



Obr. 3.1: Blokové schéma principu jednoduchého klasifikátoru.

4 Výsledky dosažené v literatuře

Automatické tagování hudebních děl se v posledních letech stalo díky neustálému zvyšování výpočetního výkonu a rozmachu neuronových sítí vyhledávanou výzvou pro mnoho výzkumných týmů z celého světa. V této krátké kapitole jsou shrnuty dosavadní výsledky, které lze dohledat v publikovaných pracích. Zároveň nutno podotknout, že získané poznatky sloužily jako výchozí bod pro návrh architektur neuronových sítí v této práci.

Následující tabulka stručně shrnuje nejlepší dosažené výsledky během několika posledních let. Nutno podotknout, že všechny tyto publikované práce využívaly výhradně přístup založený na *neuronových sítích* a *MagnaTagATune Dataset*. Díky tomu lze jejich výsledky snadno porovnávat.

Tab. 4.1: Porovnání výsledků dosažených v literatuře pomocí neuronových sítí.

Popis	**Publikace**	**Rok**	**Vstupní data**	**ROC-AUC**
4 konvoluční vrstvy	Choi et al. [29]	2016	*mel spectrogram*	0,894
4 konvoluční a 2 GRU vrstvy	Nayyar et al. [30]	2017	*mel spectrogram*	0,887
5 GRU vrstev	Song et al. [31]	2018	*mel spectrogram*	0,794
5 GRU vrstev	Song et al. [31]	2018	*scattering transform*	0,909
4 konvoluční vrstvy	Song et al. [31]	2018	*scattering transform*	0,904

5 Návrh systému

Tato kapitola se zabývá návrhem systému pro automatické tagování hudebních děl. Pozornost je postupně věnována použitým technologiím, výběru a předzpracování vhodného datasetu a v poslední řadě pak návrhu a trénování architektur neuronových sítí ve funkci *multi-label klasifikátoru*.

Veškeré zdrojové soubory, které byly pro účely této práce vytvořeny, lze nalézt na přiloženém DVD (viz příloha C). Podrobnější dokumentaci ke zdrojovým souborům včetně návodu k jejich spuštění lze získat v příloze B. Z hlediska zdrojových souborů je tato práce rovněž publikována jako GitHub repozitář[1].

5.1 Použité technologie

Tato kapitola stručně shrnuje a popisuje prostředky zvolené pro realizaci systému z hlediska software i hardware.

5.1.1 Python

K realizaci celého systému byl zvolen programovací jazyk *Python* ve verzi 3.7.7 pro lokální zpracování dat a ve verzi 3.6.2 pro zpracování dat v rámci *MetaCentrum VO* (více viz kap. 5.1.2). Jedná se o vysokoúrovňový multiplatformní skriptovací programovací jazyk, který je vyvíjen jako *open source* a v současné době je znám jako jeden z nejpopulárnějších programovacích jazyků zejména ve vědecké a výzkumné komunitě. Python byl pro řešení této práce zvolen především z důvodu jednoduché a názorné syntaxe a snadné práce s neuronovými sítěmi za použití vhodných knihoven. Právě využití knihoven je pro programování v jazyce Python stěžejní.

SciPy

Jako *SciPy*[2] je označován ekosystém několika knihoven pro Python, které jsou zaměřeny na vědeckou práci s různými typy dat. Nejpodstatnějšími jsou zejména knihovny pro práci s datovými poli *NumPy* [32] a *pandas* [33]. Za zmínění pak stojí i knihovna *Matplotlib* [34], která slouží především k vykreslování grafických závislostí.

Librosa

Librosa[3] je knihovnou sdružující nástroje pro analýzu zvukových souborů. Obsahuje základní stavební bloky pro výstavbu systémů zaměřených na získávání hudebních informací.

[1] `https://github.com/renesemela/masters-thesis-music-autotagging`
[2] `https://scipy.org/`
[3] `https://librosa.github.io/`

TensorFlow

TensorFlow[4] je knihovna vyvíjená pod hlavičkou společnosti *Google*, díky které jsou algoritmicky pokryta takřka všechna odvětví strojového učení. Hlavní předností TensorFlow jsou široké možnosti využití hardwarových prostředků. Výpočty lze provozovat na CPU i GPU v takřka libovolném počtu. Výpočty na GPU jsou zde pak silně spjaty s technologií pro paralelní výpočty *CUDA*[5]. Popis všech těchto technologií však široce překračuje rámce této práce a podrobnější informace lze získat např. v [16, 35].

Keras

Keras[6] je pak pouze vysokoúrovňové API pro práci s TensorFlow, díky kterému lze využívat jednodušší a názornější syntaxi při práci s neuronovými sítěmi. Mimo TensorFlow umí Keras pracovat i nad jinými knihovnami (např. *Theano* nebo *Deeplearning4j*).

scikit-learn a scikit-multilearn

Obě tyto knihovny jsou výkonnými nástroji pro práci se strojovým učením. V rámci této práce byly z těchto knihoven využívány především evaluační algoritmy pro klasifikátory a dále pak algoritmy pro rozdělení datasetů.

Značným přínosem pro tuto práci byla zejména knihovna *scikit-multilearn*, která obsahuje speciálně upravený algoritmus pro pseudonáhodné rozdělení datasetu s respektem k rovnocennému zastoupení jednotlivých tříd (funkce `iterative_train_test_split`[7]).

Další knihovny

Mezi další použité knihovny patřila HTTP knihovna *Requests*, dále pak knihovna *beautifulsoup4*, která zde sloužila k procházení zdrojových kódů webových stránek. V další řadě pak byly použity jednoduché knihovny jako např. *sqlite3* pro práci s databázemi nebo *datetime* pro zjišťování aktuálního času a mnoho dalších.

5.1.2 MetaCentrum VO

MetaCentrum VO[8] je virtuální organizací, která je určena pro akademickou obec a je provozována pod hlavičkou sdružení *CESNET*. Tato organizace poskytuje bezplatný datový a výpočetní prostor akademickým pracovníkům a studentům pro řešení jejich akademických prací. V principu se jedná o možnost převést své vlastní výpočty do části výpočetní infrastruktury, která je MetaCentrem spravována. Obrovskou předností využití těchto služeb je přístup k hardwarovým prostředkům o vysokém výpočetním výkonu, které by jinak byly pro běžného uživatele nedosažitelné.

[4] `https://www.tensorflow.org/`
[5] `https://developer.nvidia.com/cuda-zone`
[6] `https://keras.io/`
[7] `http://scikit.ml/stratification.html`
[8] `https://metavo.metacentrum.cz/`

5.1.3 Výpočetní prostředky UTKO FEKT VUT

Při řešení této práce byly využity hardwarové prostředky *Ústavu telekomunikací Fakulty elektrotechniky a komunikačních technologií Vysokého učení technického v Brně*. Jedná se zejména o GPU *NVIDIA GeForce RTX 2080 Ti 11 GB* a *NVIDIA GeForce RTX 2080 8 GB*. Díky takto výkonnému hardware bylo možné zpracovat následující kapitoly ve značně širší míře, než by bylo možné na běžně dostupném hardware.

5.2 MagnaTagATune Dataset

MagnaTagATune Dataset[9] vznikl jako výsledek práce [36], která se zabývala novými možnostmi získávání obsahově založených dat na základě her. V principu se jednalo o formu *kolaborativního tagování* hudby, které prováděla uživatelská základna této hry.

Tento dataset se stal velice populárním ve většině prací zabývajících se automatickým tagováním hudebních děl (viz např. [29, 30, 31]) zejména z důvodu volně přístupných zvukových dat pod licencí *Creative Commons BY-NC-SA*[10], což je u hudebních dat zpravidla velmi výjimečný jev. Právě z tohoto důvodu byl tento dataset vybrán i pro řešení této práce.

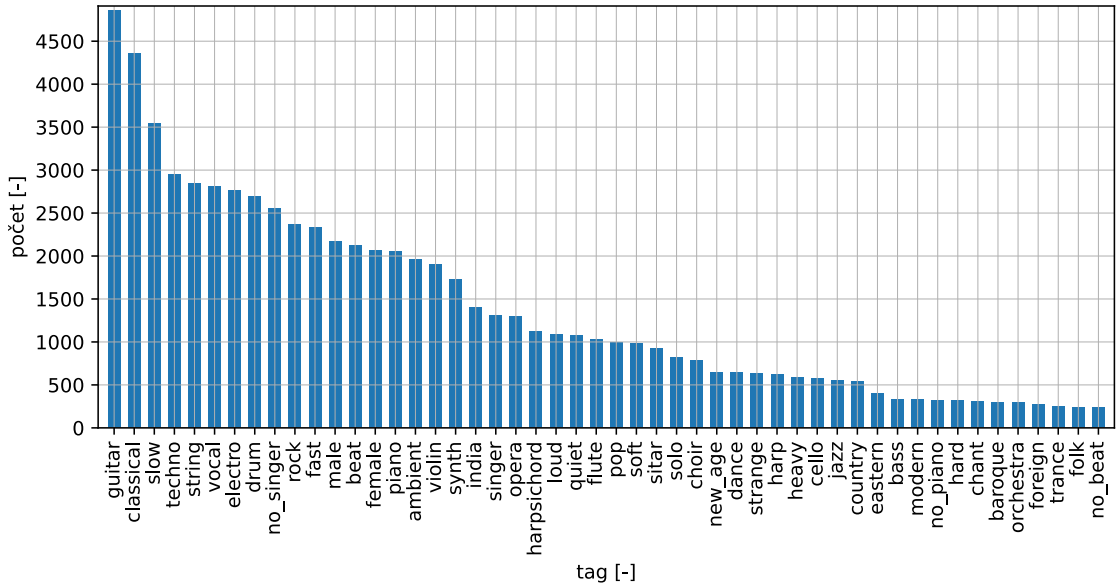
Celkem je v datasetu obsaženo 25 863 hudebních klipů o délce 29,1 s, které vznikly segmantací celkového počtu 5 405 skladeb. Formátem dat je jednokanálová MP3 o vzorkovací frekvenci 16 kHz a přenosové rychlosti 32 kbps. Ke každému klipu jsou pak přiřazeny konkrétní tagy v CSV souboru pomocí tzv. *one-hot kódování*. Celkový počet unikátních tagů v rámci datasetu je pak 188.

Hlavním problémem datasetu je v první řadě jeho nevyváženost, kdy je nejfrekventovanější tag zastoupen celkem 4861krát a např. 40. nejfrekventovanější pak pouze 337krát. Druhým problémem je pak přítomnost synonym mezi jednotlivými tagy. Tento problém je však na rozdíl od nevyváženosti snadno řešitelný.

5.2.1 Předzpracování datasetu

V první řadě byla pomocí jednoduchého slovníku vzájemně sloučena synonyma, a tím byl objem tagů v datasetu zmenšen ze 188 na 134. Z tohoto počtu bylo následně vybráno 50 nejfrekventovanějších tagů, které byly určeny jako třídy pro trénování klasifikátoru (viz obr. 5.1).

[9] `http://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset`
[10] `https://creativecommons.org/licenses/by-nc-sa/1.0/legalcode`



Obr. 5.1: Zastoupení jednotlivých tagů v rámci MagnaTagATune Dataset.

Veškerá zvuková data pak byla v rámci předzpracování převedena z formátu MP3 na formát WAV o bitové hloubce 16 a vzorkovací frekvenci 12 kHz. Tento převod s sebou přinesl nemalé zvětšení objemu zvukových dat, nicméně WAV formát je stěžejní pro snadnou manipulaci v rámci programovacího jazyka Python. Během konverze byly odhaleny 3 porušené soubory, které byly z datasetu odstraněny.

Celý dataset byl následně co nejvíce náhodně rozdělen na trénovací, validační a testovací množiny v poměru 70 : 15 : 15, respektive 18 102 : 3 879 : 3 879. Náhodné rozdělení probíhalo s respektem k rovnocennému zastoupení všech tagů.

5.3 Vstupní data

Jako realizace vstupních dat pro klasifikátor byl vybrán *mel spektrogram* (viz kap. 1.1.1). Jeho vhodnost k tomuto účelu byla potvrzena např. v [29, 30]. Parametry použitého mel spektrogramu byly zvoleny následovně:

- *počet mel pásem*: 96,
- *délka časových rámců*: 512,
- *překryv časových rámců*: 50 %,
- *typ okna*: Hann,
- *délka okna*: 512,
- *délka FFT*: 512.

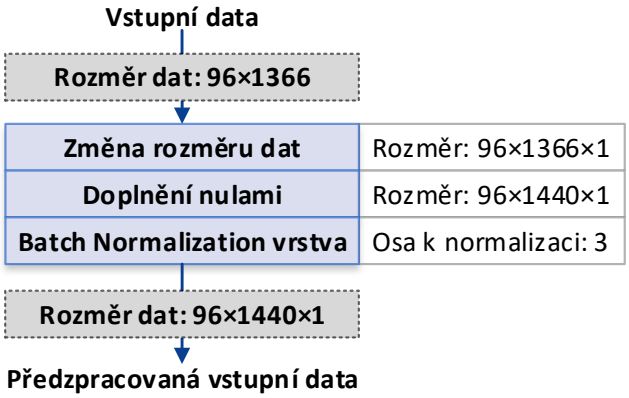
Výsledkem tohoto nastavení potom byla data o rozměrech 96×1366 pro každý hudební klip. Vzhledem k vysokým paměťovým nárokům při zpracování této formy dat byla pro trénování neuronových sítí implementována funkce dávkového zpracování s velikostí 32 vstupních souborů na dávku.

5.4 Architektury neuronových sítí a jejich trénování

Jak již byl nastíněno v kapitole 4, trendem současné doby je pro automatické tagování hudebních děl využití *neuronových sítí*, a tradiční strojové učení tak zůstává prakticky zcela nevyužito. Neuronové sítě v tomto odvětví zároveň dosahují vysokých kvalit a s každou přibývající prací na toto téma je výsledná kvalita posouvána již pouze v řádech setin až desetin procent.

V citované literatuře z kap. 4 lze pozorovat využití základních technik pro konstrukci konvolučních neuronových sítí, které jsou volitelně kombinovány se zpětnovazebními prvky. Na základě těchto poznatků byly navrženy architektury neuronových sítí v této kapitole. Veškeré zde navržené architektury zároveň vycházejí i ze série experimentů a rovněž byla využita některá nestandardní řešení za účelem dosažení co nejlepších výsledků.

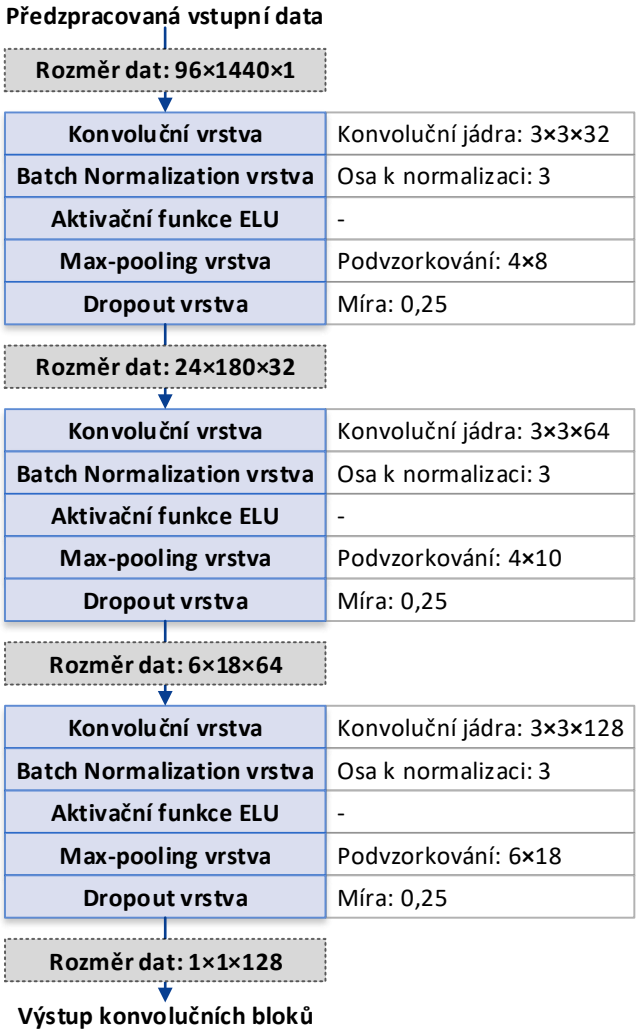
Všechny architektury navržené v této kapitole sdílí několik charakteristik. První takovou charakteristikou je *blok předzpracovávající vstupní data*, který je znázorněn na obr. 5.2. Úkolem první vrstvy tohoto bloku je formální získání třetího rozměru u dvourozměrných dat za účelem dalšího zpracovávání v síti. Druhá vrstva má potom za úkol doplnit nulami okraje dat tak, aby byl získán lépe dělitelný rozměr pro další zpracovávání. Batch Normalization vrstva následně provádí normalizaci vstupních dat v rámci jednotlivých dávek dat.



Obr. 5.2: Ukázka bloku předzpracování vstupních dat neuronové sítě.

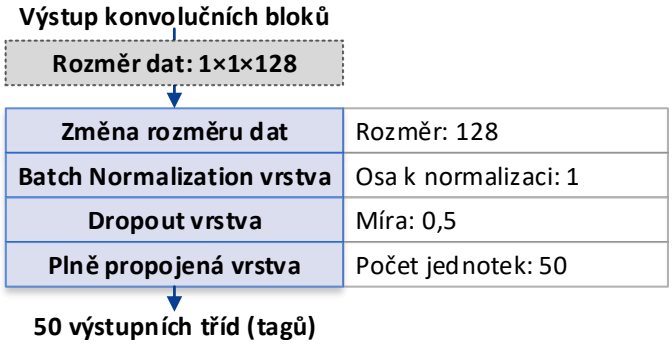
Pro konstrukci neuronových sítí v této práci byl zvolen základní *konvoluční blok*, který je složen z *konvoluční vrstvy*, *Batch Normalization vrstvy*, *aktivační funkce typu ELU*, *pooling vrstvy typu max-pooling* a *dropout vrstvy*. Základním principem tohoto bloku je průchod dat konvoluční vrstvou, kde díky aplikaci konvolučních jader dochází k navyšování třetího rozměru dat. Batch Normalization vrstva následně zajišťuje normalizaci dat před vstupem do aktivační funkce a pooling vrstva pak zmenšuje výstupní rozměr dat. Dropout vrstva je následně zařazena za účelem vyvarování se přeučení sítě. Skládání podobných konvolučních bloků za sebe je běžnou praxí při návrhu neuronových sítí (viz [29, 30, 31]). Postupně zde dochází ke zmenšování základního rozměru dat a zároveň

dochází i k navyšování třetího rozměru až do krajního případu $1 \times 1 \times n$, kde n značí počet konvolučních jader konvoluční vrstvy posledního bloku. Na obr. 5.3 lze pozorovat názorný příklad třívrstvé konvoluční sítě včetně rozměrů dat a jejich změny při průchodu konvolučními bloky.

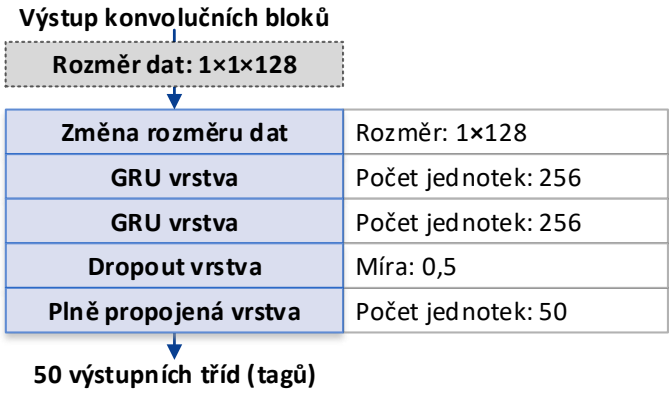


Obr. 5.3: Ukázka spojování konvolučních bloků a jejich vlivu na rozměr dat.

Následně byla navržena dvě různá zakončení neuronových sítí, kterými je daná síť klasifikována buď jako *plně konvoluční* (anglicky *Fully Convolutional Neural Network*, zkráceně FCNN), nebo jako *konvolučně-rekurentní* (anglicky *Convolutional Recurrent Neural Network*, zkráceně CRNN). Tyto *konečné bloky* lze pozorovat na obr. 5.4. Zařazení Batch Normalization vrstvy před výstupní vrstvu u plně konvoluční architektury se osvědčilo jako vhodný nástroj k redukci nebo alespoň k oddálení přeučení sítě i přesto, že se nejedná o standardní použití této vrstvy.



(a) Plně konvoluční síť.



(b) Konvolučně-rekurentní síť.

Obr. 5.4: Ukázka různých zakončení navržených neuronových sítí.

Pro všechny architektury v této práci byl využit optimalizační algoritmus *Adam* s parametrem *learning rate* nastaveným na hodnotu 0,0001. Jako ztrátová funkce byla využita *binární vzájemná entropie* a sledovanými hodnotami při trénování byly: hodnota *ztrátové funkce*, *přesnost* a hodnota *ROC-AUC*. Jako prevence proti přeučení sítě byl implementován algoritmus *předčasného zastavení* (anglicky *early stopping*), jehož úkolem bylo sledovat vývoj validační hodnoty ROC-AUC a zastavit trénování v případě, kdy už nedocházelo k dalšímu zlepšování této hodnoty. Tento algoritmus obsahuje parametr *trpělivosti* (anglicky *patience*), kterým je určeno, kolik trénovacích epoch algoritmus vyčkává na zlepšení před zastavením trénování. Z důvodu kolísání validační hodnoty ROC-AUC byl tento parametr nastaven na 20 epoch.

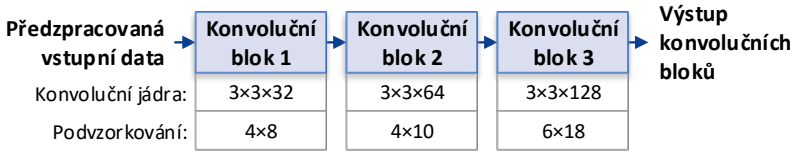
Během návrhu byla věnována zvýšená pozornost optimálnímu nastavení parametru learning rate algoritmu Adam, který je v praxi velmi často nastavován experimentálně. Příliš malá hodnota způsobuje pomalé trénování a hodnota validační ztrátové funkce příliš pomalu konverguje ke svému minimu. Z tohoto důvodu je potřeba síť podrobit většímu počtu trénovacích *epoch*, přičemž epochou se rozumí případ, kdy při trénovacím procesu projdou sítí všechna trénovací data jedenkrát. Příliš vysoká hodnota parametru learning rate pak způsobuje náhodné chování validační ztrátové funkce a ztráta zpravidla s rostoucím počtem epoch stoupá.

Nutno zároveň poznamenat, že trénování neuronových sítí probíhá do jisté míry náhodným principem (počáteční inicializace vah, pořadí vstupních dat apod.), kdy může ve specifických případech docházet k abnormálně dobrým nebo abnormálně špatným výsledkům. V praxi je tato náhodnost realizována generátorem pseudonáhodných čísel, který je ve výchozím nastavení velmi často inicializován systémovým časem. Proto bylo z důvodu reprodukovatelnosti výsledků v této práci využito tzv. *random seed*, což je číselný parametr, který je využit jako inicializátor generátoru pseudonáhodných čísel místo systémového času. Díky stejné inicializaci generátoru jsou pak veškerá generovaná čísla generována stejně při jakémkoli běhu programu. Veškeré výsledky v této kapitole byly získány při nastavení random seed na hodnotu 0 (pokud není řečeno jinak).

Jednotlivé navržené architektury neuronových sítí v této práci vycházejí ze základního návrhu plně konvoluční architektury a konvolučně-rekurentní architektury a prakticky se od sebe liší pouze počtem použitých konvolučních bloků a s tím spojeným počtem konvolučních filtrů a mírou podvzorkování v jednotlivých blocích. Z tohoto důvodu jsou schémata jednotlivých architektur zjednodušena a konkrétní rozepsané architektury lze najít vždy v příslušných přílohách.

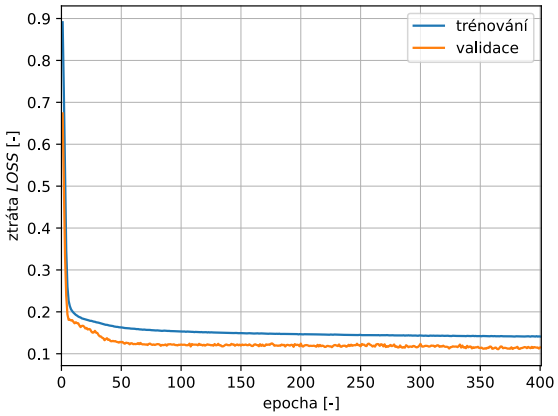
5.4.1 FCNN3 (plně konvoluční neuronová síť, 3 vrstvy)

První navrženou architekturou je třívrstvá plně konvoluční neuronová síť, ve které jsou předzpracovaná vstupní data (viz obr. 5.2) postupně přivedena na vstupy konvolučních bloků (viz obr. 5.5), jejichž výstup je přiveden na zakončení plně konvoluční neuronové sítě (viz obr. 5.4a).

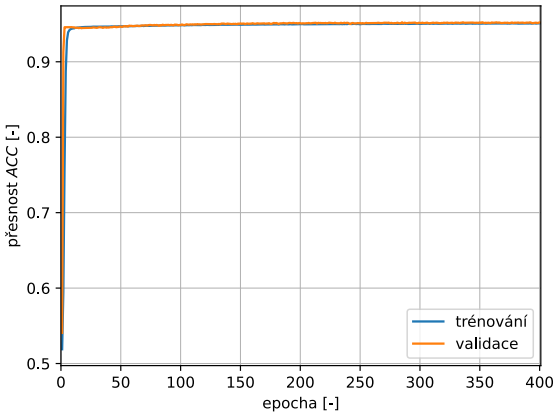


Obr. 5.5: Navržená architektura sítě (3 vrstvy).

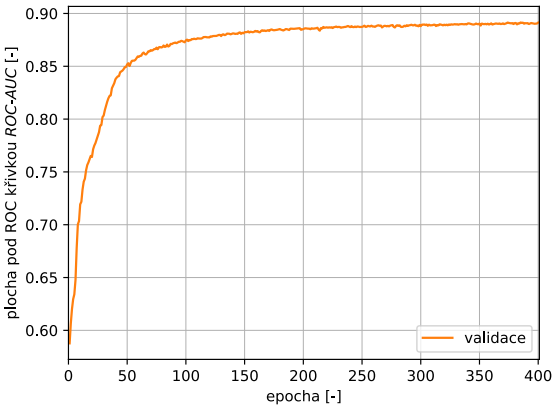
Již pohledem na míru podvzorkování, kterou je potřeba během tří konvolučních bloků provést, aby bylo dosaženo cíleného rozměru dat na jejich výstupu, lze předpokládat, že síť bude pravděpodobně potřeba podrobit vysokému počtu trénovacích epoch. Zároveň zde hrozí riziko, že malá komplexnost sítě zapříčiní neuspokojivé výsledky.



(a) Ztráta.



(b) Přesnost.



(c) ROC-AUC.

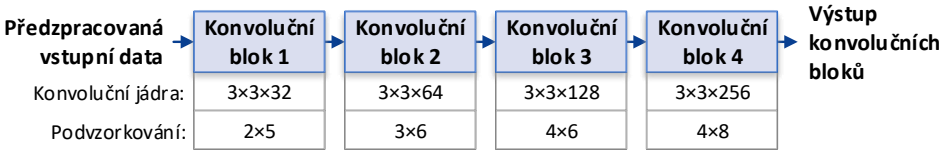
Obr. 5.6: Průběh trénování architektury FCNN3 (MagnaTagATune Dataset, 50 tagů).

Na obr. 5.6 lze pozorovat průběh trénování této sítě. Je patrné, že hodnota validační ztrátové funkce opravdu konvergovala velmi pomalu ke svému minimu, což platí rovněž i pro hodnotu ROC-AUC, která konvergovala velmi pomalu ke svému maximu.

Podrobněji rozepsanou architekturu této neuronové sítě lze nalézt v tab. A.1.

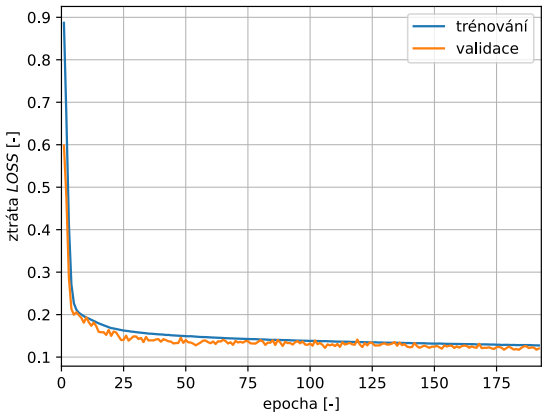
5.4.2 FCNN4 (plně konvoluční neuronová síť, 4 vrstvy)

Druhá navržená architektura přidává další konvoluční blok, čímž je zajištěno pozvolnější podvzorkování dat a zároveň je zvýšena komplexnost celé sítě v důsledku zvýšení počtu trénovatelných parametrů.

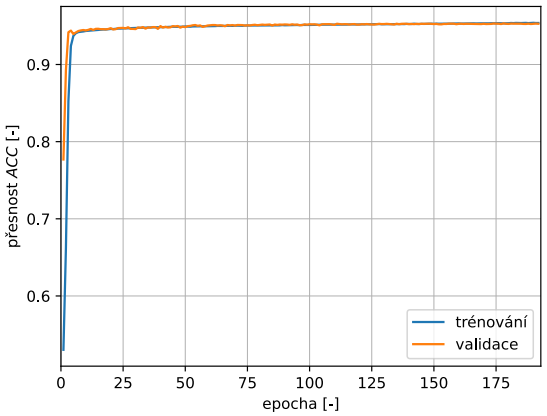


Obr. 5.7: Navržená architektura sítě (4 vrstvy).

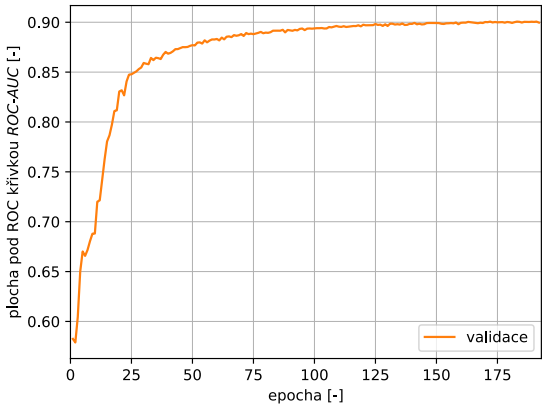
Na obr. 5.8 lze pozorovat, jaký vliv mělo zvýšení komplexnosti sítě. Zejména počet epoch nutných pro konvergenci k maximu hodnoty ROC-AUC na validačních datech se značně zmenšil. Zároveň bylo dosaženo znatelně vyšší maximální hodnoty ROC-AUC.



(a) Ztráta.



(b) Přesnost.



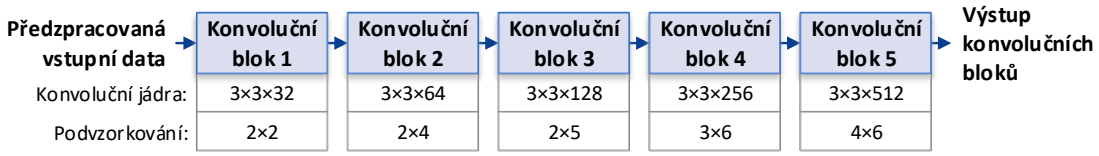
(c) ROC-AUC.

Obr. 5.8: Průběh trénování architektury FCNN4 (MagnaTagATune Dataset, 50 tagů).

Podrobněji rozepsanou architekturu této neuronové sítě lze nalézt v tab. A.2.

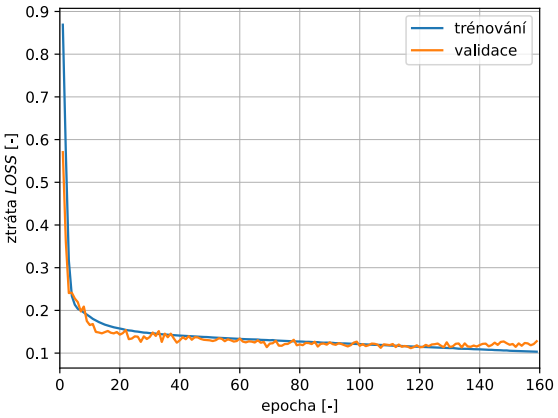
5.4.3 FCNN5 (plně konvoluční neuronová síť, 5 vrstev)

V pořadí třetí architektura přidává další konvoluční blok podle obr. 5.9.

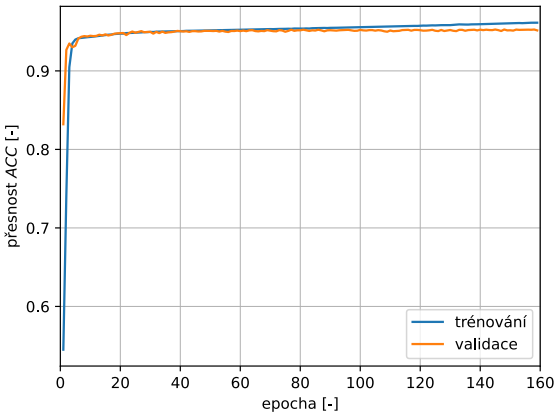


Obr. 5.9: Navržená architektura sítě (5 vrstev).

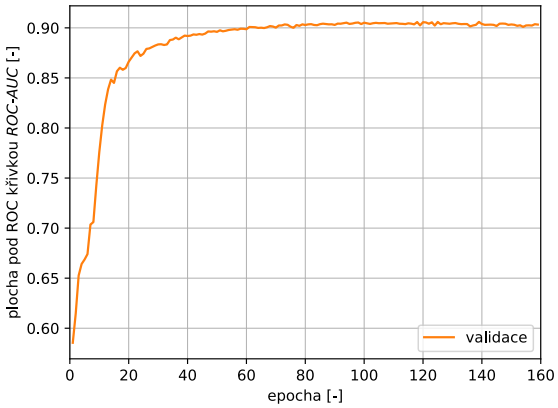
Na obr. 5.10 lze opět pozorovat průběh trénování této neuronové sítě. Opět došlo ke zmenšení potřebného počtu epoch ke konvergenci sledovaných průběhů, avšak přibližně od 120. epochy lze pozorovat náznak přeučení sítě, kdy se hodnota validační ztrátové funkce začala zvětšovat a hodnota ROC-AUC na validačních datech začala po svém maximu opět klesat.



(a) Ztráta.



(b) Přesnost.



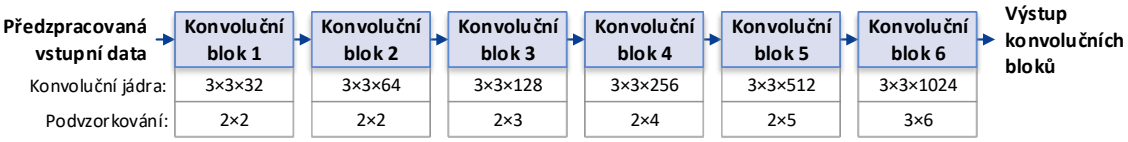
(c) ROC-AUC.

Obr. 5.10: Průběh trénování architektury FCNN5 (MagnaTagATune Dataset, 50 tagů).

Podrobněji rozepsanou architekturu této neuronové sítě lze nalézt v tab. A.3.

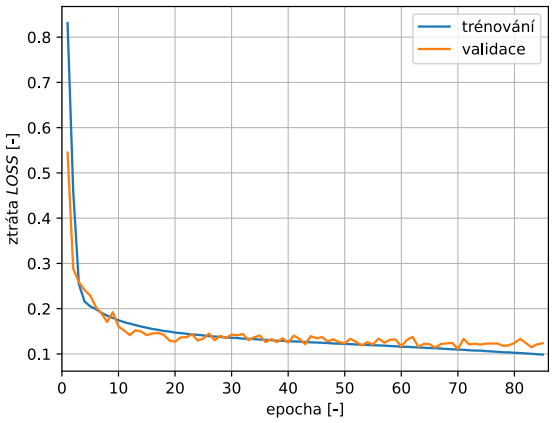
5.4.4 FCNN6 (plně konvoluční neuronová síť, 6 vrstev)

Poslední plně konvoluční architektura obsahuje 6 konvolučních bloků podle obr. 5.11.

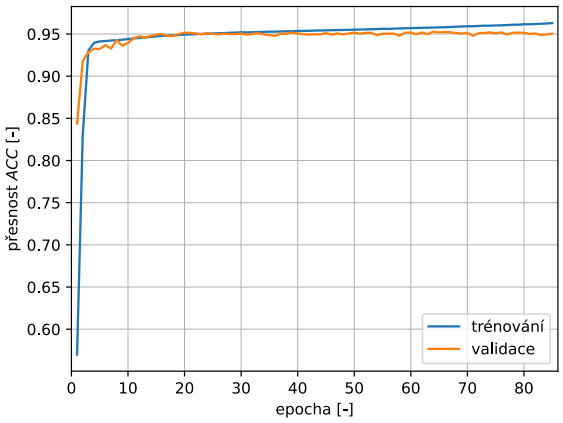


Obr. 5.11: Navržená architektura sítě (6 vrstev).

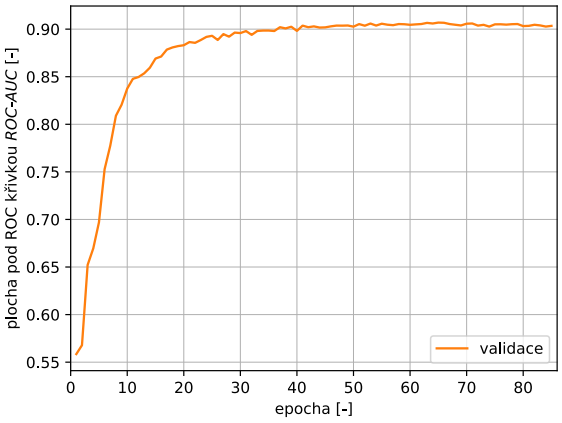
Rovněž i u této sítě se objevil náznak přeučení okolo 70. epochy, avšak do tohoto bodu byla konvergence sítě nejrychlejší ze všech dosavadních architektur.



(a) Ztráta.



(b) Přesnost.



(c) ROC-AUC.

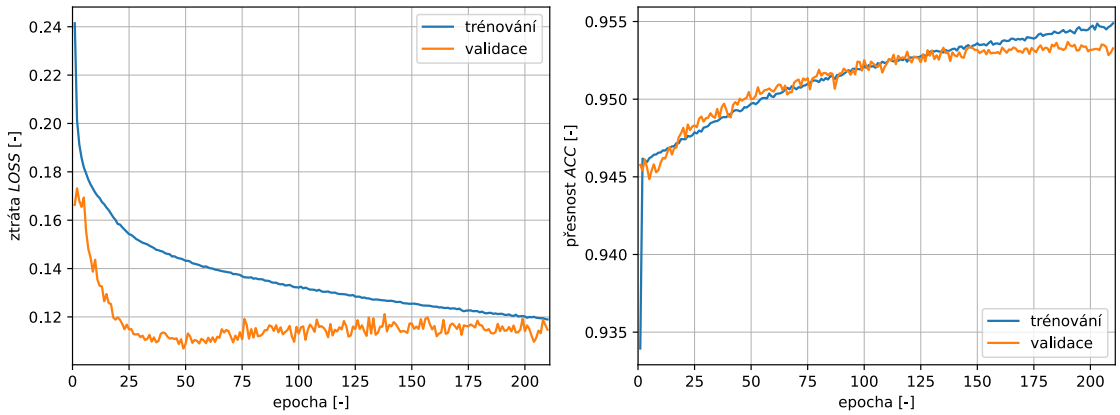
Obr. 5.12: Průběh trénování architektury FCNN6 (MagnaTagATune Dataset, 50 tagů).

Podrobněji rozepsanou architekturu této neuronové sítě lze nalézt v tab. A.4.

5.4.5 CRNN3 (konvolučně-rekurentní neuronová síť, 3 vrstvy)

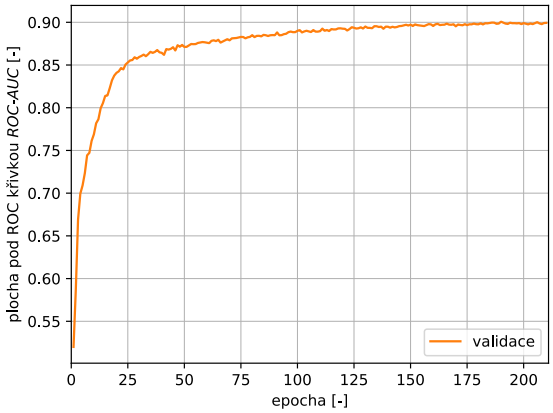
Všechny konvolučně-rekurentní architektury využívají stejné uspořádání konvolučních bloků jako jejich příbuzné plně konvoluční architektury. Jediným rozdílem je využití rekurentního zakončení sítě podle obr. 5.4b. Tato konvolučně-rekurentní architektura je tedy složena z bloku předzpracování (viz obr. 5.2), konvolučních bloků třívrstvé plně konvoluční sítě (viz obr. 5.5) a rekurentního zakončení sítě (viz obr. 5.4b).

Na obr. 5.13 lze pozorovat, že tato architektura i přes neideální průběh hodnoty ztrátové funkce vykazovala na validačních datech vyšší hodnotu ROC-AUC při takřka polovičním počtu trénovacích epoch než plně konvoluční architektura se třemi vrstvami.



(a) Ztráta.

(b) Přesnost.



(c) ROC-AUC.

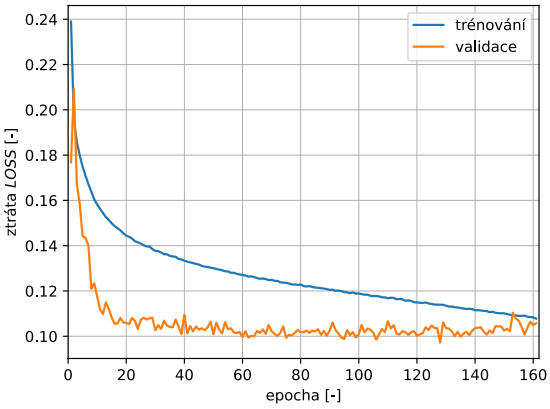
Obr. 5.13: Průběh trénování architektury CRNN3 (MagnaTagATune Dataset, 50 tagů).

Podrobněji rozepsanou architekturu této neuronové sítě lze nalézt v tab. A.5.

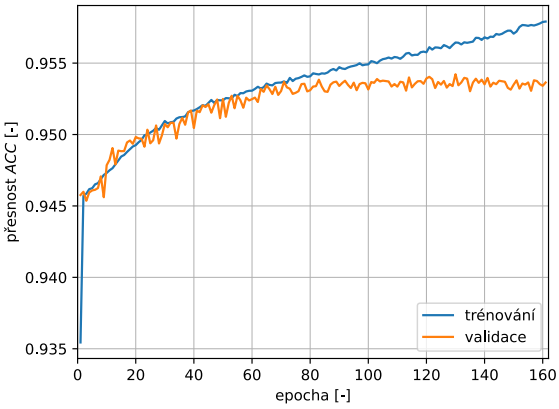
5.4.6 CRNN4 (konvolučně-rekurentní neuronová síť, 4 vrstvy)

Rovněž i tato architektura vychází konfigurací konvolučních bloků z příbuzné čtyřvrstvé plně konvoluční varianty a přidává rekurentní zakončení sítě. I zde lze pozorovat konvergenci k maximu hodnoty ROC-AUC na validačních datech v menším počtu trénovacích epoch než u obdobné čtyřvrstvé plně konvoluční sítě. Maximální hodnota ROC-AUC byla i u této architektury vyšší než u obdobné plně konvoluční varianty.

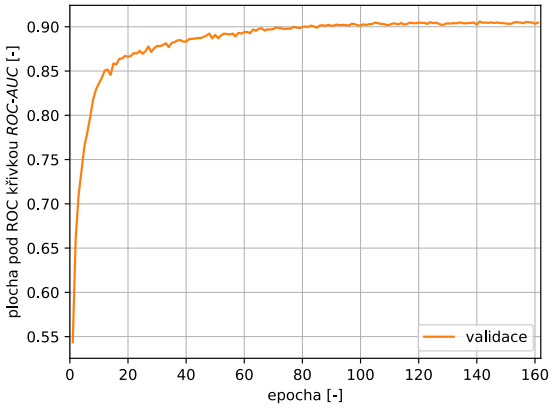
I zde se mezi 120. a 140. epochou se objevilo přeučení sítě, kdy všechny sledované hodnoty začaly vykazovat odklon od požadovaných průběhů.



(a) Ztráta.



(b) Přesnost.



(c) ROC-AUC.

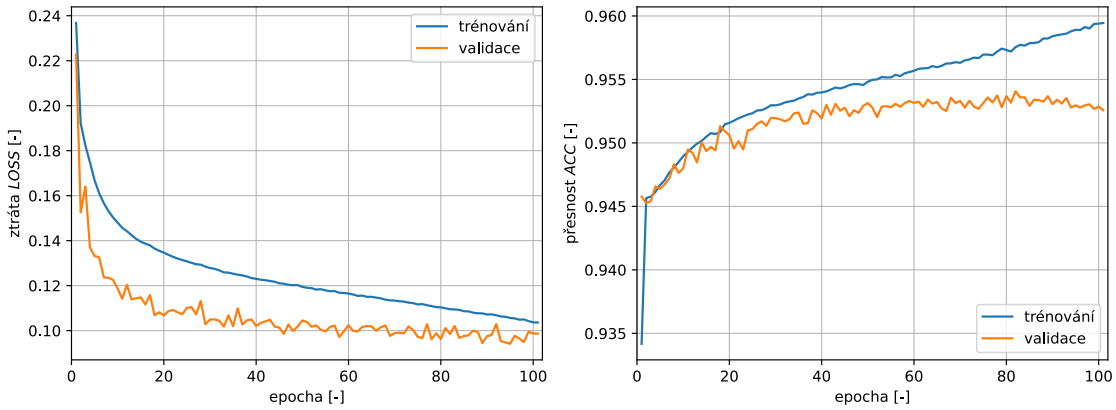
Obr. 5.14: Průběh trénování architektury CRNN4 (MagnaTagATune Dataset, 50 tagů).

Podrobněji rozepsanou architekturu této neuronové sítě lze nalézt v tab. A.6.

5.4.7 CRNN5 (konvolučně-rekurentní neuronová síť, 5 vrstev)

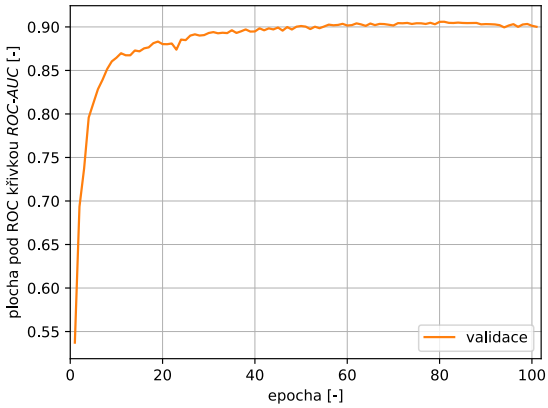
Také pětivrstvá konvolučně-rekurentní architektura se svými výsledky nesla v podobném duchu jako její předchůdci. Oproti obdobné pětivrstvé plně konvoluční architektuře došlo ke konvergenci k maximální hodně ROC-AUC na validačních datech v menším počtu trénovacích epoch. Maximum této hodnoty zůstalo v tomto případě podobné jako maximum plně konvoluční varianty.

Okolo 80. epochy se však začalo vyskytovat přeučení sítě, kdy přestalo docházet ke zlepšování hodnoty ROC-AUC na validačních datech a rovněž začala klesat i validační přesnost.



(a) Ztráta.

(b) Přesnost.



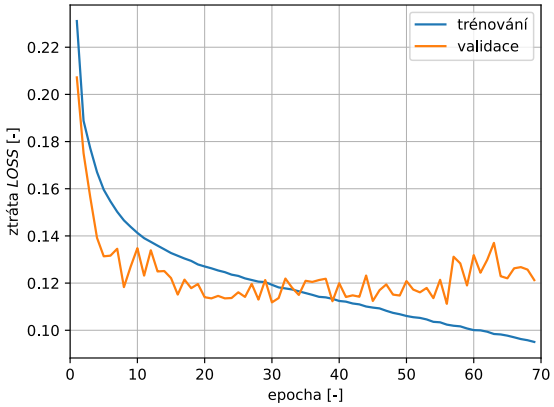
(c) ROC-AUC.

Obr. 5.15: Průběh trénování architektury CRNN5 (MagnaTagATune Dataset, 50 tagů).

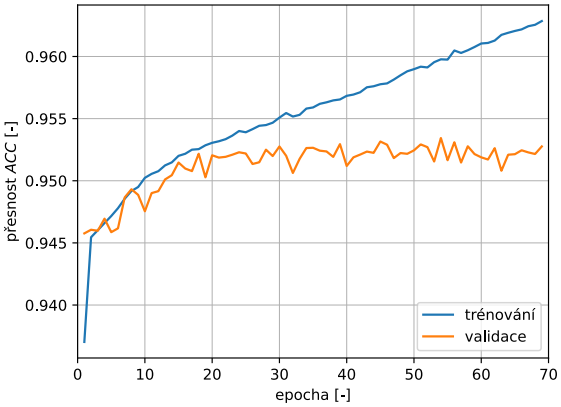
Podrobněji rozepsanou architekturu této neuronové sítě lze nalézt v tab. A.7.

5.4.8 CRNN6 (konvolučně-rekurentní neuronová síť, 6 vrstev)

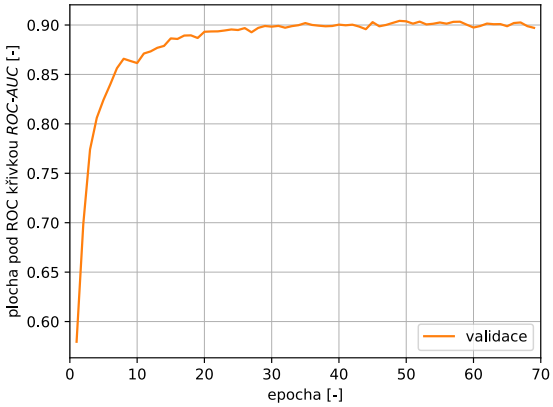
Také poslední konvolučně-rekurentní architektura vykazovala během trénování již typické chování svých konvolučně-rekurentních předchůdců. I zde se však okolo 50. epochy vyskytlo přeučení sítě.



(a) Ztráta.



(b) Přesnost.



(c) ROC-AUC.

Obr. 5.16: Průběh trénování architektury CRNN6 (MagnaTagATune Dataset, 50 tagů).

Podrobněji rozepsanou architekturu této neuronové sítě lze nalézt v tab. A.8.

5.4.9 Shrnutí trénování

Všechny architektury až na třívrstvou plně konvoluční neuronovou síť (FCNN3) vykazovaly vysoké hodnoty ROC-AUC na validačních datech a takřka vždy překročily hodnotu 0,9. Horší výsledky trénování u třívrstvé plně konvoluční architektury (FCNN3) lze přikládat především malé komplexnosti sítě a příliš vysokému podvzorkování mezi jednotlivými konvolučními bloky sítě.

Přirozeným jevem u všech architektur byl klesající počet trénovacích epoch nutných pro dosažení maxima validační hodnoty ROC-AUC při zvyšování komplexnosti sítě. Nutno však pamatovat, že se zvyšováním komplexnosti sítě dochází i ke zvyšování počtu parametrů uvnitř sítě a s tím jsou spjaty rovněž i vyšší časové nároky na provedení jedné trénovací epochy.

Počet parametrů, čas nutný k provedení jedné epochy, počet epoch nutných pro dokončení trénování a celkový čas potřebný k provedení trénování jsou uvedeny v tab. 5.1. Pro zvýšení výpovědní hodnoty byly hodnoty v tabulce získány statistickým zpracováním deseti různých trénování architektur s deseti různými random seed. Výsledky jsou uvedeny ve tvaru *průměr* ± *směrodatná odchylka*. Časy epoch byly u všech architektur získány na jednotném GPU NVIDIA GeForce RTX 2080 Ti 11 GB (viz kap. 5.1.3).

Tab. 5.1: Shrnutí trénování navržených architektur neuronových sítí.

Architektura	**Počet parametrů**	**Počet epoch**	**Čas epochy**	**Výsledný čas**
FCNN3	100 534	349 ± 42	69 s	6,69 ± 0,81 h
FCNN4	403 638	208 ± 23	82 s	4,73 ± 0,53 h
FCNN5	1 599 670	130 ± 12	116 s	4,18 ± 0,37 h
FCNN6	6 351 030	81 ± 6	135 s	3,03 ± 0,22 h
CRNN3	796 086	243 ± 25	70 s	4,72 ± 0,48 h
CRNN4	1 190 582	162 ± 13	82 s	3,70 ± 0,30 h
CRNN5	2 569 398	106 ± 14	116 s	3,42 ± 0,44 h
CRNN6	7 686 326	65 ± 9	135 s	**2,45 ± 0,33 h**

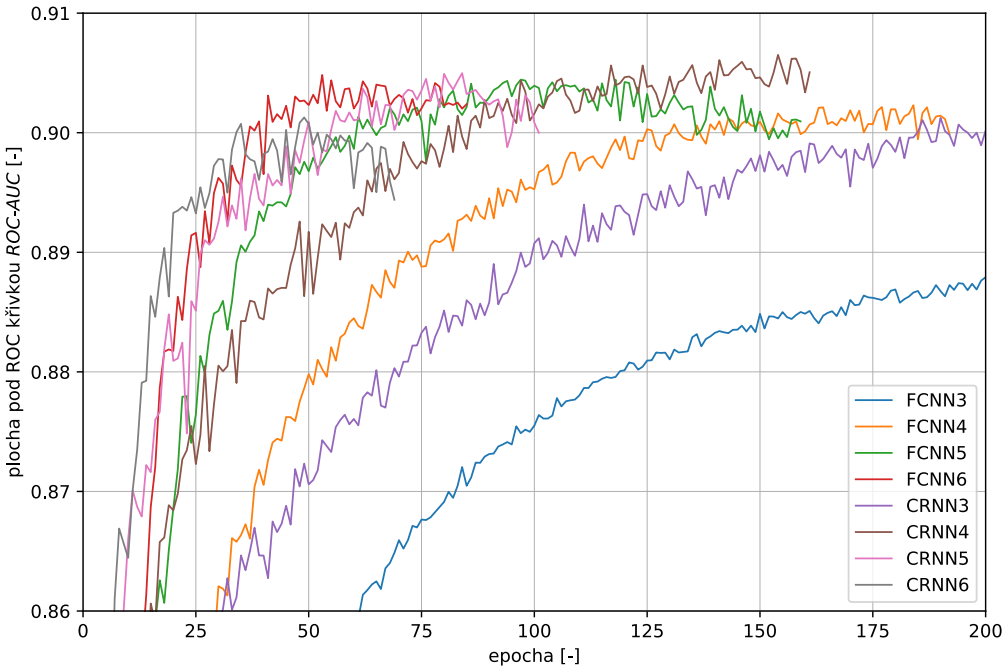
V tab. 5.1 lze pozorovat, že i se zvyšováním komplexnosti jednotlivých sítí a s tím spojeným vyšším časem, který byl potřebný k provedení jedné epochy, docházelo ke snižování celkového času, který byl nutný k dokončení trénování. Z tohoto hlediska jasně zvítězila *konvolučně-rekurentní architektura se šesti vrstvami* (CRNN6).

Rozdíl v řádech hodin pro natrénování několika různých architektur lze považovat za zanedbatelný. Při větším objemu dat, kdy by se tento rozdíl vyšplhal na řády dní, by již tato skutečnost začala na významu podstatně nabývat.

6 Testování a evaluace

V kap. 5.4.9 byly předběžně shrnuty výsledky trénování všech navržených architektur neuronových sítí. Podkladem pro toto shrnutí byla zejména validační data využívaná během trénování. Účelem této kapitoly je podrobit všechny architektury testování na testovací množině dat. Jedná se o data, která jsou pro síť absolutně neznámá (anglicky *unseen data*) a byla z datasetu odebrána při jeho rozdělení během předzpracování.

Na obr. 6.1 lze v jednom grafu pozorovat průběhy hodnot ROC-AUC na testovacích datech pro všechny architektury. Pro názornost byla použita pouze výseč, která co nejlépe zobrazuje rozdíly v konvergenci hodnot ROC-AUC u jednotlivých architektur.



Obr. 6.1: Průběhy hodnot ROC-AUC na testovacích datech pro všechny architektury.

Z obr. 6.1 lze usoudit, že skrze svou rychlou konvergenci a vysoké ROC-AUC je jedním z favoritů šestivrstvá plně konvoluční architektura (FCNN6 – *červená křivka*). Dalším možným kandidátem je skrze nejvyšší dosaženou hodnotu ROC-AUC i čtyřvrstvá konvolučně-rekurentní architektura (CRNN4 – *hnědá křivka*) a v těsném závěsu za ní potom pětivrstvá konvolučně-rekurentní architektura (CRNN5 – *růžová křivka*).

Lze říci, že primárním záměrem je dosažení co nejvyšší hodnoty ROC-AUC na neznámých datech. Nelze však pronášet konečné závěry na základě jednoho trénování pro každou síť, protože vlivem specifické pseudonáhodné inicializace jednotlivých sítí může tato hodnota vykazovat nezanedbatelné výkyvy. Z tohoto důvodu bylo do tab. 6.1 statisticky zpracováno deset různých trénování všech architektur s deseti různými random seed. Výsledky jsou uvedeny ve tvaru *průměr* ± *směrodatná odchylka*.

Tab. 6.1: Maximální hodnoty ROC-AUC na testovacích datech pro všechny architektury.

Architektura	**Maximální ROC-AUC**
FCNN3	0,8899 ± 0,0024
FCNN4	0,9012 ± 0,0014
FCNN5	0,9036 ± 0,0007
FCNN6	0,9035 ± 0,0008
CRNN3	0,9021 ± 0,0012
CRNN4	**0,9046 ± 0,0016**
CRNN5	0,9042 ± 0,0008
CRNN6	0,9004 ± 0,0009

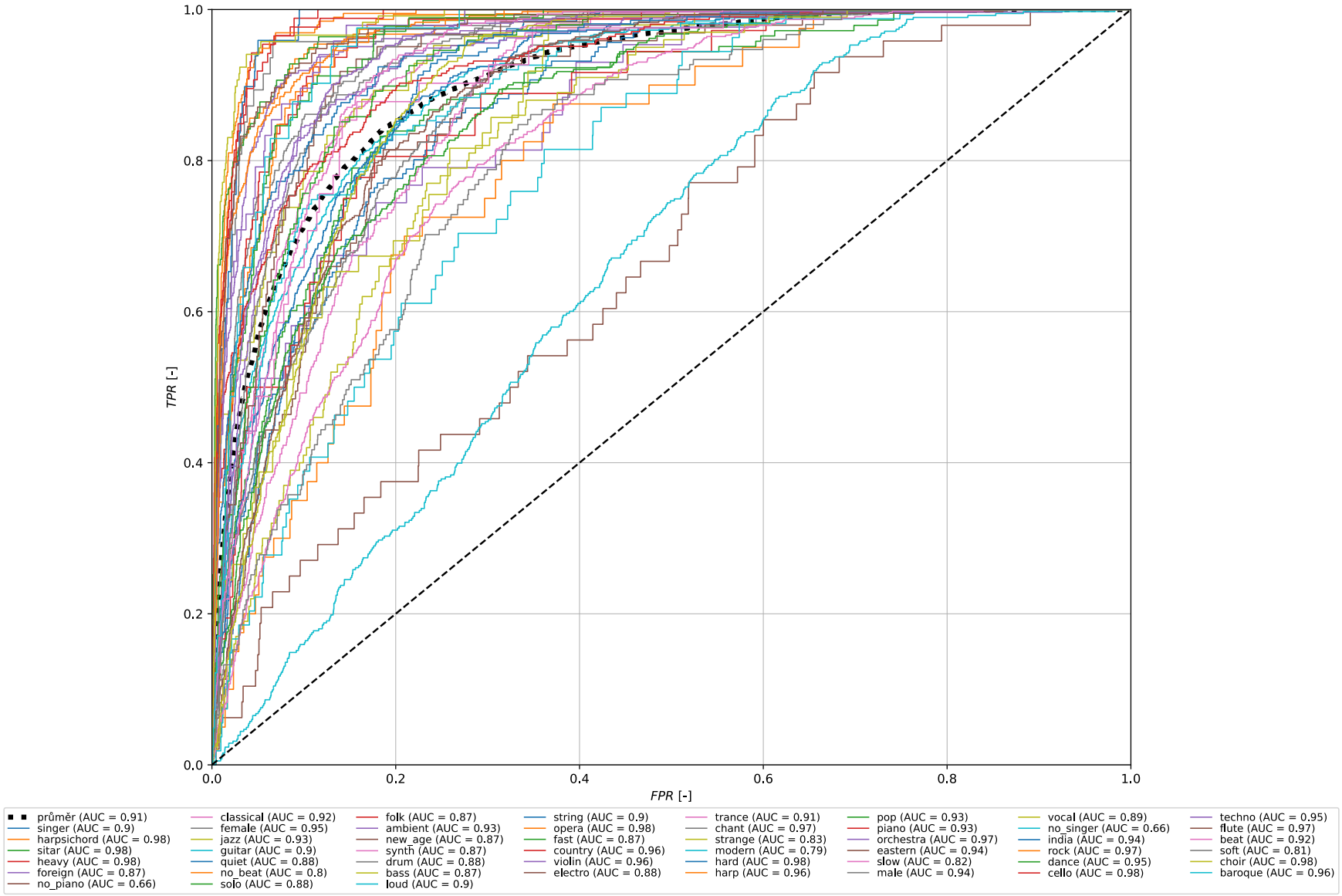
Na základě tab. 6.1 lze vyvodit závěr, že nejlepších výsledků dosahovala *čtyřvrstvá konvolučně-rekurentní architektura* (CRNN4) a ve velmi těsném závěsu za ní potom její pětivrstvá alternativa (CRNN5). Tyto dvě architektury zároveň podle tab. 5.1 nevykazovaly žádné podstatnější rozdíly v časových nárocích na trénování, tudíž se následující texty zabývají především čtyřvrstvou konvolučně-rekurentní architekturou. Zároveň nutno podotknout, že taktéž i ostatní architektury byly ve většině případů svými výsledky na velmi dobré úrovni.

6.1 CRNN4 (konvolučně-rekurentní neuronová síť, 4 vrstvy)

Vzhledem k faktu, že tato architektura dosahovala během testování nejlepších výsledků, je vhodné podrobněji přiblížit její charakteristiky. Na obr. 6.2 lze pozorovat ROC křivky pro jednotlivé tagy, které byly neuronovou sítí klasifikovány. Rozdílná hodnota ROC-AUC u průměrné ROC křivky je zde dána faktem, že tyto ROC křivky jsou zobrazeny pouze pro jedno konkrétní trénování sítě, avšak v tab. 6.1 je statisticky zpracováno těchto trénování více.

Pohledem na ROC křivky lze snadno odhalit dva tagy, jejichž natrénování takřka úplně selhalo. Konkrétně se jedná o tagy *no_piano* (*hnědá křivka*) a *no_singer* (*tyrkysová křivka*). Důvodem k nenatrénování může být zejména fakt, že se z hudebního hlediska jedná o velmi obecné tagy, u kterých se zřejmě nepodařilo nalézt dostatek rozlišovacích znaků pro správnou klasifikaci. Tag *no_piano* je zároveň v datasetu zastoupen velmi málo. Zajímavým faktem je, že oba tyto tagy jsou téměř jako jediné (na rozdíl od ostatních tagů) záporně definované. Tag *no_beat* byl na druhou stranu natrénován s řádově vyšším ROC-AUC a jeho zastoupení v datasetu je ještě menší než je tomu u tagu *no_piano*. Zde však lze z obsahového hlediska předpokládat dobrou rozlišitelnost od ostatních tagů.

Lze předpokládat určité zvýšení hodnoty ROC-AUC při odstranění těchto problematických tagů, což může být vhodné zejména pro systémy automatického tagování používané v praxi.



Obr. 6.2: ROC křivky jednotlivých tagů pro architekturu CRNN4 (MagnaTagATune Dataset, 50 tagů, epocha 154)

6.2 Shrnutí testování a evaluace

Testovaní navržených neuronových sítí ukázalo, že takřka všechny architektury vykazovaly velmi uspokojivé hodnoty ROC-AUC na testovacích datech v porovnání s výsledky jiných prací zabývajících se podobnou problematikou (viz kap. 4).

Lze říci, že primárním měřítkem kvality architektur v této práci je nejvyšší dosažená hodnota ROC-AUC na testovacích datech a z tohoto hlediska zvítězila *čtyřvrstvá konvolučně-rekurentní architektura* (CRNN4) s hodnotou **0,9046** $\pm$ **0,0016**.

V tomto okamžiku je zároveň vhodné zmínit, že těchto výsledků síť dosáhla s počtem 1 190 582 parametrů. Vedle toho např. práce [31] dosáhla ROC-AUC = 0,904 s počtem 17 365 000 parametrů při čtyřvrstvé plně konvoluční architektuře, avšak za použití jiného typu vstupních dat. Stejná práce pak dosáhla ROC-AUC = 0,909 s počtem 7 776 000 parametrů při pětivrstvé kompletně rekurentní architektuře a rovněž při použití jiného typu vstupních dat. Práce [29] pak dosáhla ROC-AUC = 0,894 při čtyřvrstvé plně konvoluční architektuře a vstupními daty ve formě mel spektrogramu. Nicméně i tato práce využívala značně přeparametrizovanou architekturu neuronové sítě. Z výsledků této práce tedy vyplývá, že lze dosáhnout podobných, nebo dokonce i lepších výsledků s řádově nižším počtem parametrů neuronových sítí. Dobré výsledky byly zároveň pravděpodobně dosaženy i z důvodu věnování zvýšené pozornosti správnému předzpracování celého datasetu.

Vzhledem k dlouhodobě podobným výsledkům při použití MagnaTagATune Dataset napříč světovými pracemi lze usoudit, že možnosti tohoto datasetu jsou již maximálně využity a nelze předpokládat žádné razantní zlepšení výsledků při použití podobných technik automatického tagování, jako jsou techniky využívané v této práci.

7 Další testování

Vzhledem k faktu, že možnosti MagnaTagATune Dataset, který byl využíván při návrhu architektur v kap. 5, jsou již takřka plně vyčerpány a zároveň jsou dosti omezené (nevyváženost datasetu, malý počet vstupních dat pro méně frekventované tagy apod.), je vhodné otestovat nejlepší architektury rovněž na jiném datasetu.

7.1 Last.fm Dataset 2020

V rámci této práce byl vytvořen *Last.fm Dataset 2020*. Ten svým konceptem navazuje na *Last.fm Dataset*[1] (vycházející z *Million Song Dataset*[2] [37]), který však nebylo možné skrze jeho zastaralost použít, jelikož jeho poslední aktualizace proběhla v roce 2011. Tento dataset neobsahoval hudební data pro analýzu přímo, ale využíval vazby na API (zkratka slov *Application Programming Interface*) jiných služeb, které buď v průběhu let uzavřely svou činnost, nebo uzavřely své API pro veřejnost. Koncepce tohoto datasetu je však i v současné době aktuální, jelikož *Last.fm*[3] je stále jednou z nejrozsáhlejších služeb, která se věnuje sběru uživatelských dat v oblasti poslechu hudby. Tato data mimo jiné obsahují i tagy, které jsou skladbám přiřazovány za pomoci uživatelské základny této služby.

Last.fm Dataset 2020 je rovněž publikován jako samostatný GitHub repozitář[4].

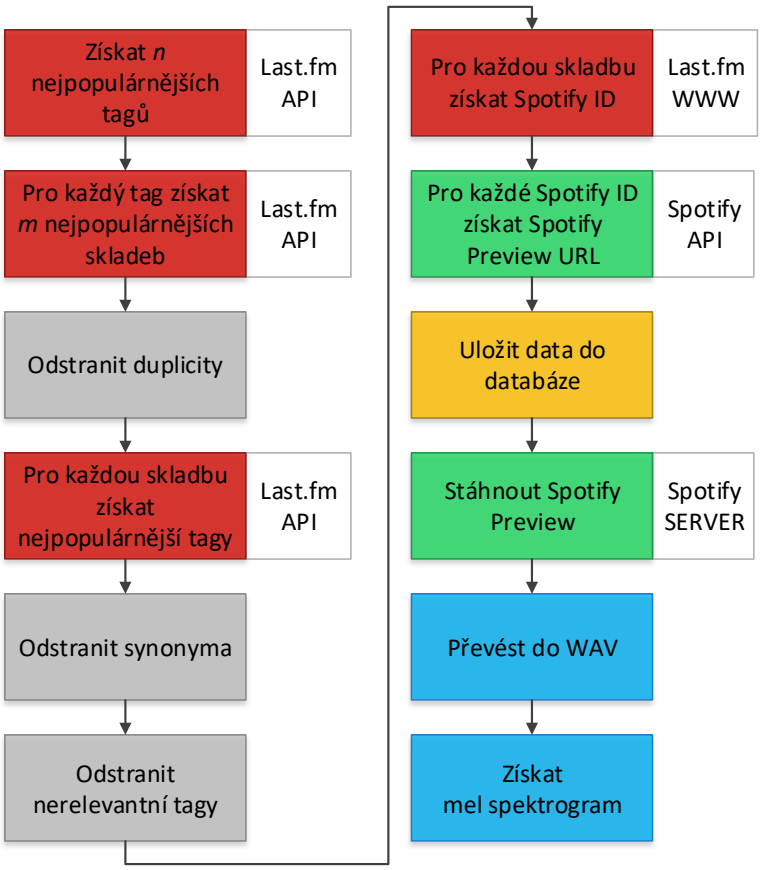
[1] `http://millionsongdataset.com/lastfm/`
[2] `http://millionsongdataset.com/`
[3] `https://www.last.fm/`
[4] `https://github.com/renesemela/lastfm-dataset-2020`

7.1.1 Sestavení databáze

K vybudování datasetu bylo využito veřejné webové API služby *Last.fm*, díky němuž bylo možné získat nejpopulárnější tagy a skladby napříč celou touto službou. Dále pak bylo API využito pro přiřazení konkrétních tagů konkrétním skladbám, a tím byla získána základní databáze datasetu. Následně pak bylo využito webové API služby *Spotify*[5] pro získání cca třicetisekundových úseků jednotlivých skladeb ve formě MP3 souborů. Blokové schéma principu sestavení datasetu lze nalézt na obr. 7.1.

V bloku odstranění synonym docházelo ke sloučení tagů se stejným významem. Příkladem mohou být například tagy *electronic* a *electronica* nebo *female_vocalist* a *female_vocalists*.

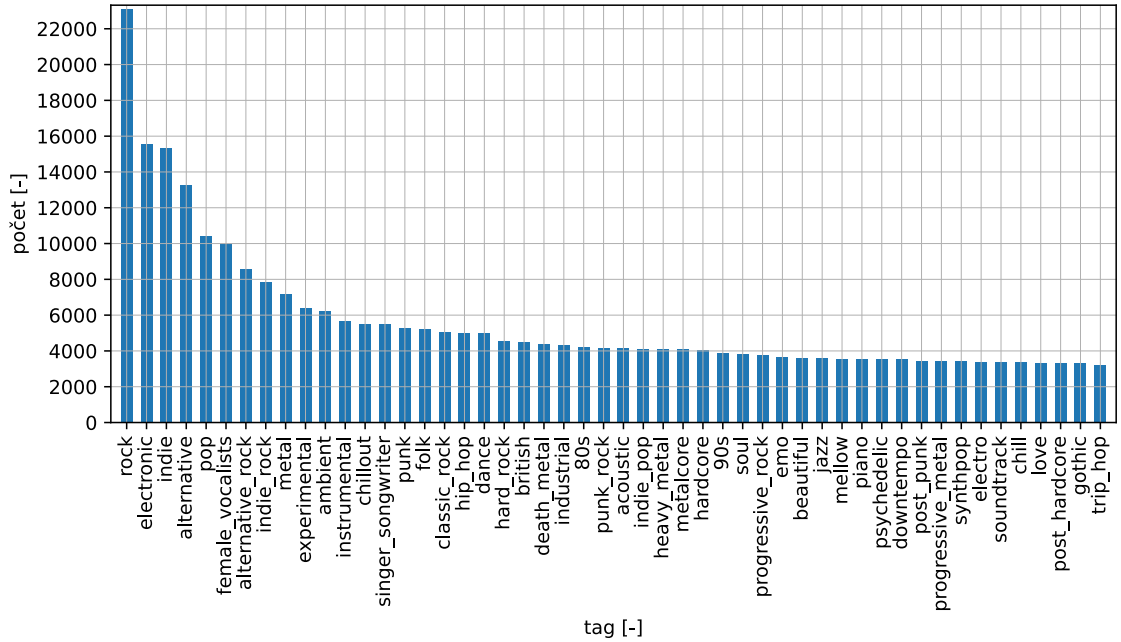
V bloku odstranění nerelevantních tagů pak docházelo k odstranění tagů, které objektivně nesouvisely s obsahem skladeb. Jednalo se například o tagy *awesome*, *albums_i_own* nebo *seen_live*.



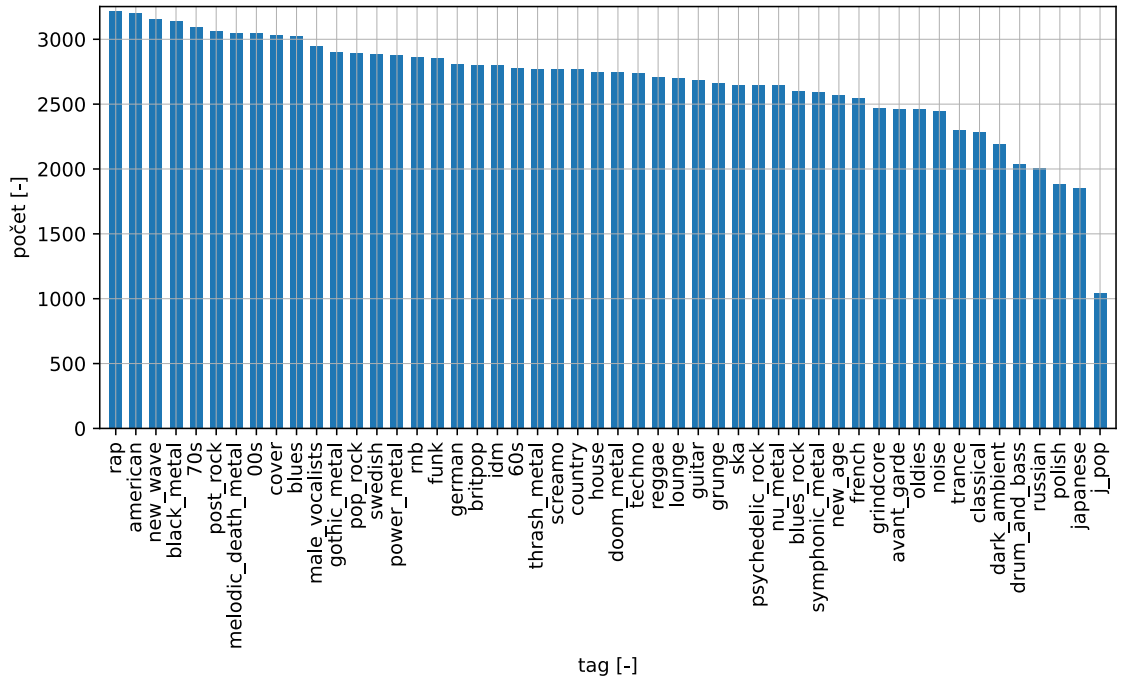
Obr. 7.1: Blokové schéma sestavení Last.fm Dataset 2020.

[5] `https://www.spotify.com/`

Pro účely testování byl sestaven dataset čítající 100 tagů a celkem 122 877 skladeb. Zastoupení jednotlivých tagů v rámci datasetu je znázorněno na obr. 7.2, kde lze pozorovat, že až na extrémy v podobě první desítky tagů a posledních několika tagů je zachována relativní vyváženost celého datasetu.



(a) Tagy 1-50.



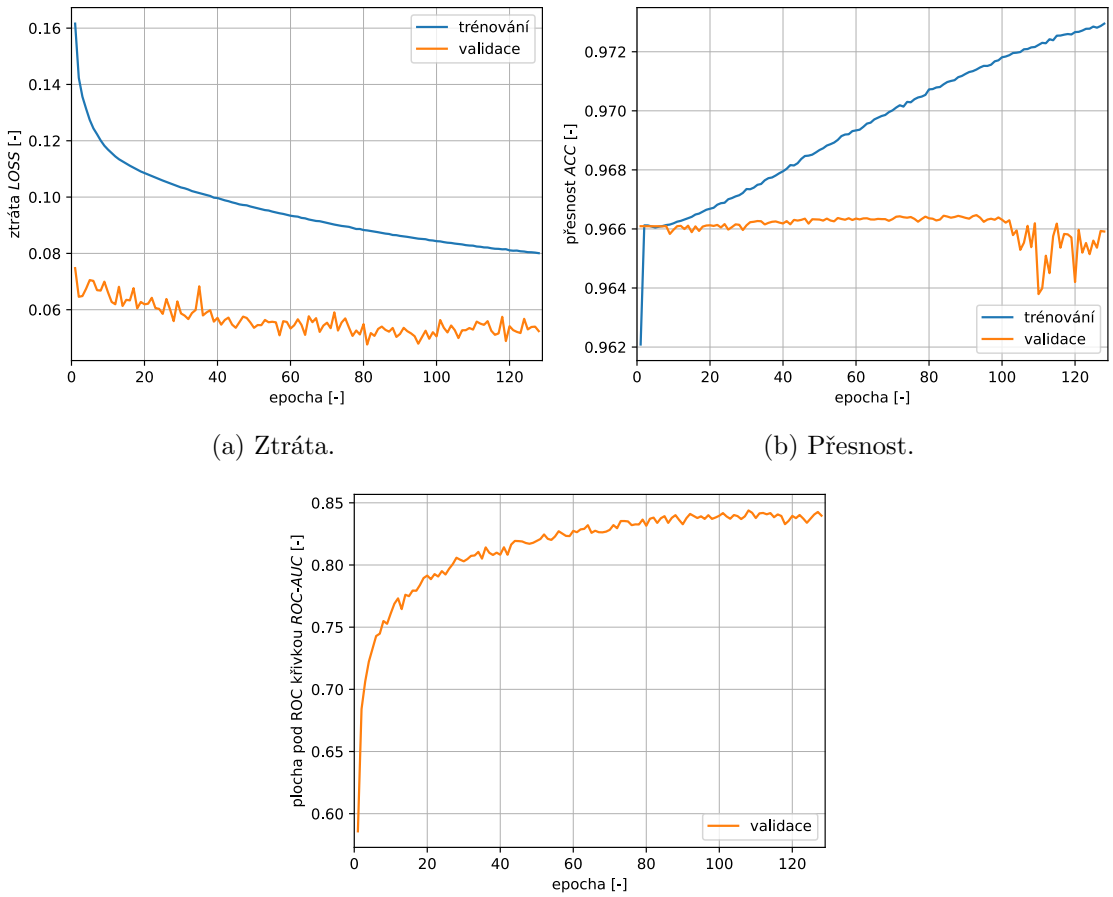
(b) Tagy 51-100.

Obr. 7.2: Zastoupení jednotlivých tagů v rámci Last.fm Dataset 2020.

7.2 Trénování

Vzhledem ke značnému zvýšení časových nároků na trénování z důvodu vyššího počtu skladeb v datasetu byly k natrénování vybrány pouze ty nejúspěšnější architektury z kapitoly 5.

Jako vzorový příklad pro zobrazení průběhu trénování byla vybrána nejúspěšnější architektura z předchozí kapitoly – *čtyřvrstvá konvolučně-rekurentní neuronová síť* (CRNN4). Tento průběh lze pozorovat na obr. 7.3.



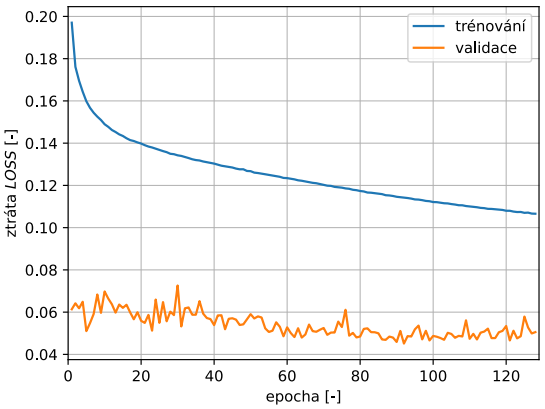
(a) Ztráta.

(b) Přesnost.

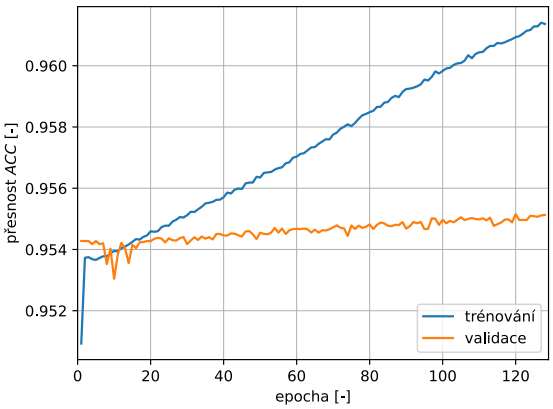
(c) ROC-AUC.

Obr. 7.3: Průběh trénování architektury CRNN4 (Last.fm Dataset 2020, 100 tagů).

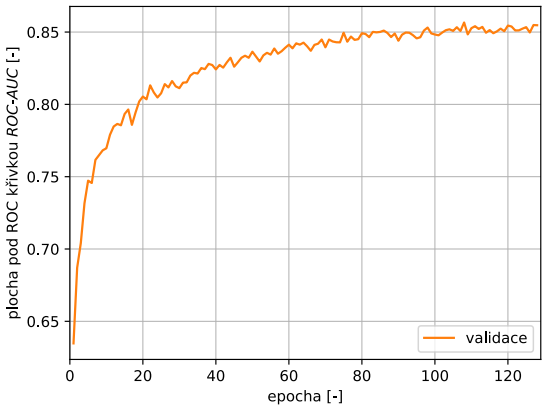
Na obr. 7.3 lze však zároveň pozorovat, že okolo 100. epochy začalo docházet k přeučení sítě a hodnota validační přesnosti začala klesat. Tento jev se podařilo eliminovat při omezení trénování pouze na 50 nejfrekventovanějších tagů datasetu (viz obr. 7.2a). Lze předpokládat, že pomocí odladění parametrů jednotlivých vrstev neuronové sítě lze předejít přeučení i s plným počtem tagů. To s sebou však přináší další nezanedbatelné časové nároky na provedení. Průběh trénování s omezením na 50 tagů lze pozorovat na obr. 7.4.



(a) Ztráta.



(b) Přesnost.

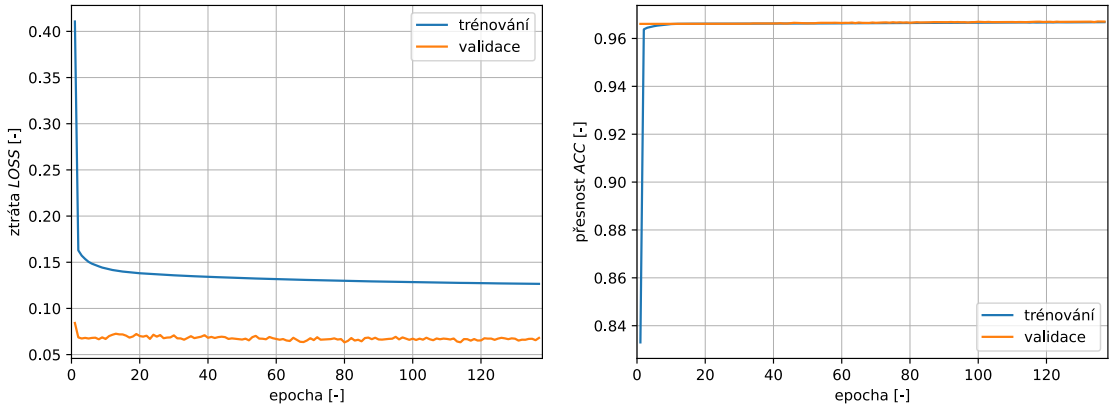


(c) ROC-AUC.

Obr. 7.4: Průběh trénování architektury CRNN4 (Last.fm Dataset 2020, 50 tagů).

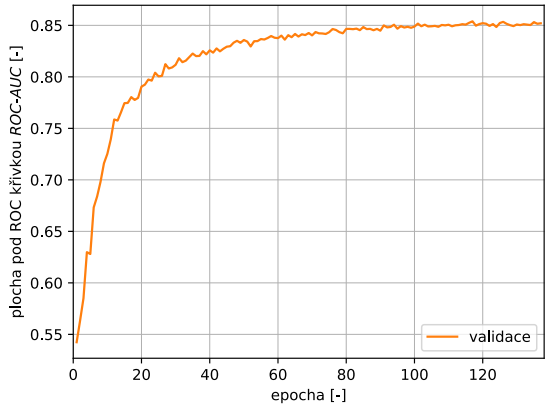
Podobný charakter průběhu trénování se objevil i u pětivrstvé konvolučně-rekurentní architektury (CRNN5). Jediným rozdílem byl fakt, že k přeučení sítě při použití plného počtu tagů docházelo při nižším počtu trénovacích epoch a validační přesnost v takovém případě kolísala a klesala výrazněji než u čtyřvrstvé architektury.

U pětivrstvé plně konvoluční architektury (FCNN5) se přeučení při použití plného počtu tagů neobjevilo vůbec a průběh trénování lze sledovat na obr. 7.5. Obdobný charakter průběhu trénování se objevil rovněž i u šestivrstvé varianty (FCNN6).



(a) Ztráta.

(b) Přesnost.

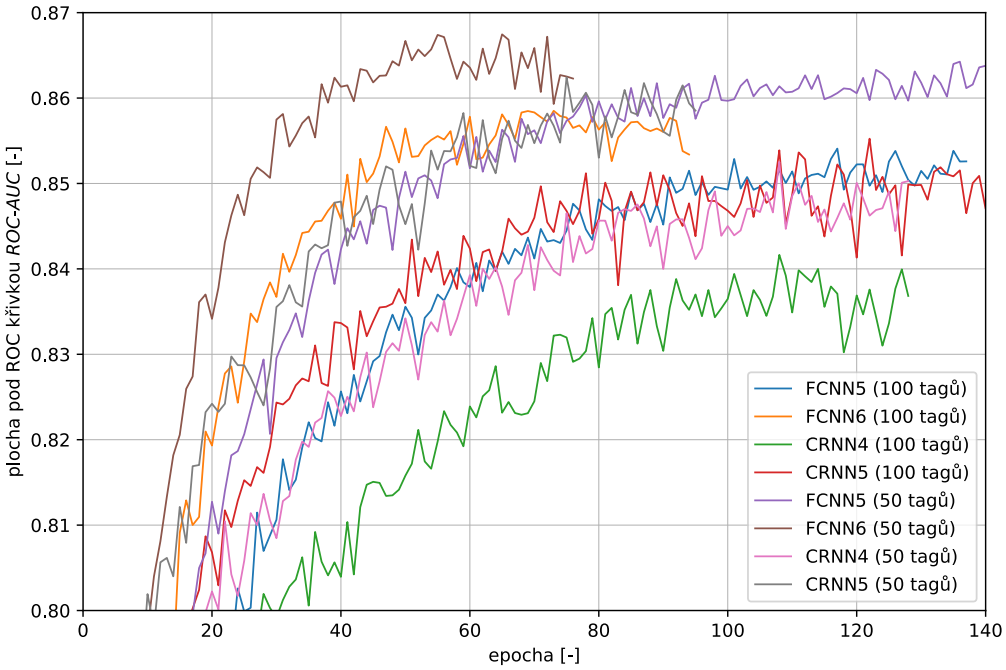


(c) ROC-AUC.

Obr. 7.5: Průběh trénování architektury FCNN5 (Last.fm Dataset 2020, 100 tagů).

7.3 Testování a evaluace

Po natrénování architektur bylo i u tohoto datasetu provedeno testování architektur na neznámých datech. Vyhodnocovacím prostředkem se stala opět hodnota ROC-AUC a na obr. 7.6 lze pozorovat průběhy těchto hodnot pro jednotlivé architektury neuronových sítí.



Obr. 7.6: Průběhy hodnot ROC-AUC na testovacích datech pro nejlepší architektury.

V tab. 7.1 jsou uvedeny nejvyšší dosažené hodnoty ROC-AUC na testovacích datech pro jednotlivé architektury. Všechny architektury byly natrénovány šestkrát za použití šesti různých random seed. Výsledky jsou uvedeny ve tvaru *průměr* ± *směrodatná odchylka*.

Tab. 7.1: Maximální hodnoty ROC-AUC na testovacích datech pro nejlepší architektury.

	50 tagů	**100 tagů**
Architektura	**Maximální ROC-AUC**	**Maximální ROC-AUC**
FCNN5	0,8644 ± 0,0018	0,8542 ± 0,0004
FCNN6	0,8675 ± 0,0012	**0,8590 ± 0,0011**
CRNN4	0,8511 ± 0,0026	0,8415 ± 0,0018
CRNN5	0,8643 ± 0,0012	0,8559 ± 0,0013

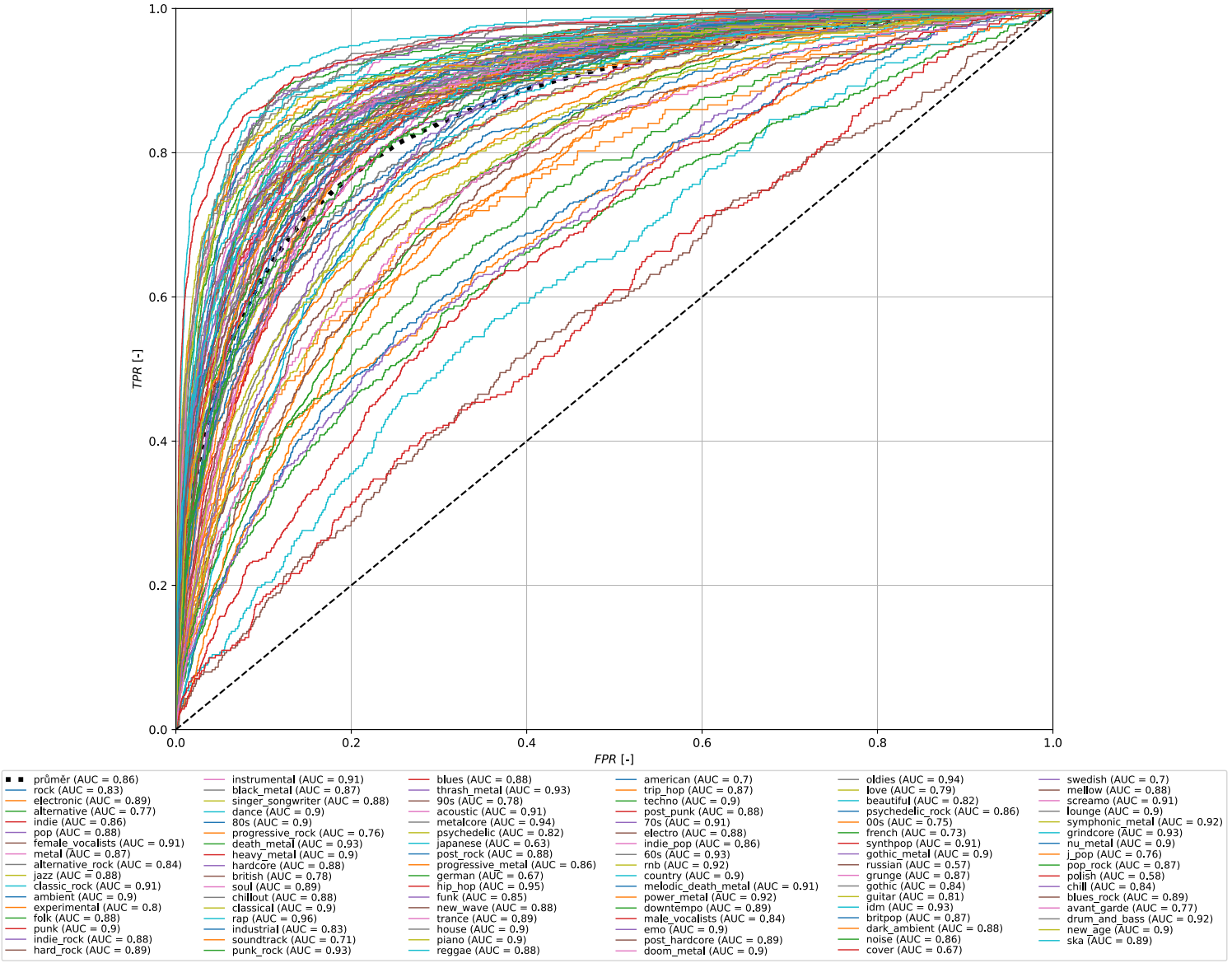
Z obr. 7.6 i tab. 7.1 lze vypozorovat, že na rozdíl od výsledků dosažených při použití MagnaTagATune Dataset se pořadí nejúspěšnějších architektur změnilo. Čtyřvrstvá konvolučně-rekurentní architektura (CRNN4) se stala nejméně úspěšnou a architektury (FCNN5 a CRNN5) vykazovaly velmi podobné výsledky. Nejlepších výsledků však dosahovala *šestivrstvá plně konvoluční architektura* (FCNN6). Lze pozorovat, že se zvyšováním

komplexnosti architektur docházelo v případě použití Last.fm Dataset 2020 ke zlepšování výsledků a lze předpokládat další zlepšení při návrhu více komplexních architektur, což může být výzvou pro navazující práce na téma automatického tagování hudebních děl.

Zároveň lze pozorovat, že omezení počtu tagů při trénování ze 100 na 50 nepřineslo v relativních rozdílech mezi architekturami žádné podstatné rozdíly. Pouze došlo k posunu všech hodnot ROC-AUC cca o 0,01 v kladném směru. Z těchto poznatků lze usuzovat, že druhá část datasetu obsahovala hůře rozlišitelné tagy, které srazily celkovou průměrnou ROC křivku a s tím i související hodnotu ROC-AUC.

Na obr. 7.7 lze pozorovat ROC křivky jednotlivých tagů pro architekturu FCNN6. Hlavní částí je zde legenda grafu, kde jsou k jednotlivým tagům přiřazeny jejich hodnoty ROC-AUC. Z těchto hodnot lze snadno dojít k následujícím závěrům:

- Nejslabších výsledků dosahovaly tagy zaměřené geograficky/národnostně (např. *british*, *german*, *japanese*, *russian*, *swedish*, *polish* a *french*). To lze přisuzovat všeobecné horší rozlišitelnosti takovýchto tagů.
- Tag *rock* na rozdíl od výsledků na *MagnaTagATune Dataset* (ROC-AUC = 0,97) dosahoval řádově horších výsledků (ROC-AUC = 0,83). To lze přisuzovat velmi obecné charakteristice tohoto tagu z hudebního hlediska. Tato obecnost se zároveň mohla přenést do služby Last.fm pomocí kolaborativního tagování uživatelskou základnou, a tím pádem i do *Last.fm Dataset 2020*.
- Některé rockové subžánry (např. *inde_rock*, *hard_rock*, *punk_rock* a další) dosahovaly oproti mateřskému žánru podstatně lepších výsledků (ROC-AUC = 0,88 až 0,93). Některé rockové subžánry (např. *alternative_rock* a *progressive_rock*) naopak dosahovaly průměrných výsledků (ROC-AUC = 0,76 až 0,84).
- Metalové, elektronické a další žánry (*instrumental*, *country*, *rap*, *hip_hop*, *rnb* a další), u kterých lze z hudební podstaty předpokládat dobrou rozlišitelnost, dosahovaly velmi dobrých výsledků. Dobrých výsledků pak zároveň dosahovaly i určité metalové subžánry, u kterých lze opět z hudebního hlediska předpokládat velmi dobrou rozlišitelnost (např. *death_metal*, *melodic_death_metal*, *metalcore*, *thrash_metal* a *symphonic_metal*).
- Tagy týkající se časového zařazení skladeb dosahovaly rozporuplných výsledků. Zatímco tagy *60s*, *70s* a *80s* dosahovaly dobrých výsledků (ROC-AUC = 0,90 až 0,93), tagy *90s* a *00s* dosahovaly řádově horších výsledků (ROC-AUC = 0,75 až 0,78). Dobrou rozlišitelnost starších hudebních období lze s největší pravděpodobností přisuzovat zejména charakteristické kvalitě zvukových záznamů v jednotlivých hudebních desetiletích (zdokonalování záznamových technologií) a rovněž pak i rychlé změně populárních žánrů v těchto obdobích. Nízkou rozlišitelnost u současnějších skladeb lze potom naopak přisuzovat relativně stálé kvalitě zvukových záznamů a široké žánrové rozmanitosti.
- U tagů, u kterých lze z hudebního hlediska předpokládat horší rozlišitelnost (*cover*, *love*, *soundtrack* a další), se opravdu špatná rozlišitelnost potvrdila (ROC-AUC = 0,67 až 0,79).

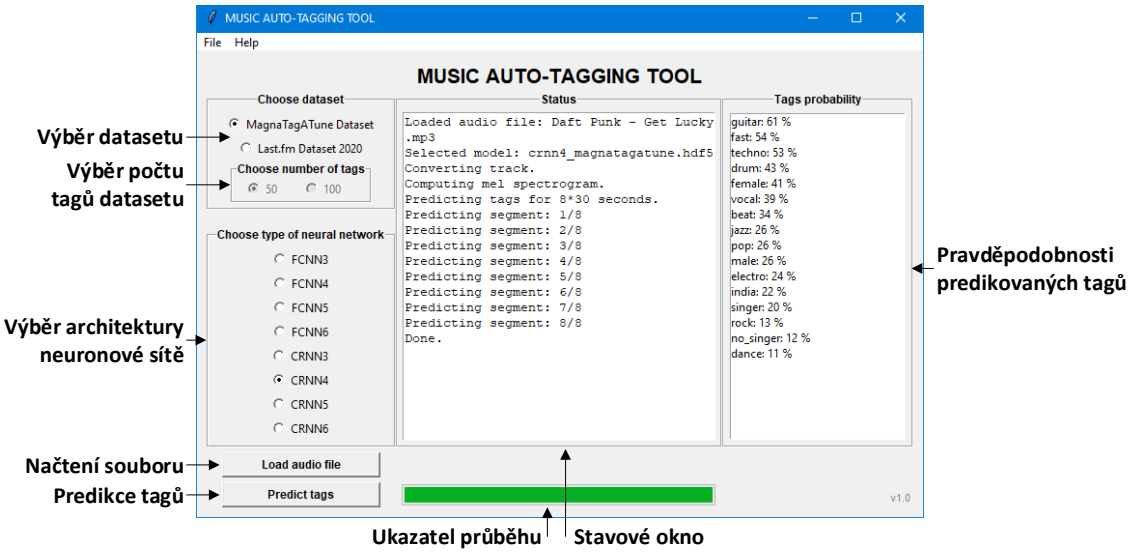


Obr. 7.7: ROC křivky jednotlivých tagů pro architekturu FCNN6 (Last.fm Dataset 2020, 100 tagů, epocha 73).

8 Aplikace pro automatické tagování hudebních děl

Pro otestování neuronových sítí na reálných datech byla v rámci této práce vyvinuta jednoduchá formulářová aplikace, která pracuje s natrénovanými modely jednotlivých architektur neuronových sítí a používá je pro predikci pravděpodobností jednotlivých tagů na uživatelem zvoleném zvukovém souboru ve formátu MP3 nebo WAV. Pro vytvoření aplikace byla využita knihovna *tkinter* pro Python.

Na obr. 8.1 lze pozorovat GUI vytvořené aplikace včetně popisků jednotlivých prvků. Ovládání aplikace lze považovat za intuitivní a nepotřebuje bližšího komentáře. Bližší informace ke korektnímu spuštění aplikace jsou obsaženy v příloze B.4.



Obr. 8.1: Ovládací prvky aplikace pro automatické tagování hudebních děl.

Závěr

V této práci byly stručně shrnuty teoretické poznatky z oboru získávání hudebních informací a následně pak možnosti parametrizace hudebních děl. Dále byla podrobně přiblížena problematika strojového učení z hlediska relevance k zadání této práce. Zvýšená pozornost byla věnována zejména umělým neuronovým sítím v podobě multi-label klasifikátoru, jejich elementárním stavebním prvkům, jejich vrstvám, možnostem trénování a následným možnostem evaluace. V další části pak byla přiblížena samotná problematika systémů pro automatické tagování hudebních děl s důrazem na konkrétní techniky používané k tomuto účelu. Navazující část následně stručně přiblížila výsledky světových prací zabývajících se stejnou nebo podobnou problematikou.

V rámci praktické části této práce byly vybrány vhodné prostředky pro implementaci celého řešení (programovací jazyk Python, HW prostředky v podobě GPU, MagnaTagATune Dataset a jeho předzpracování, mel spektrogram jako forma vstupních dat). Následně bylo navrženo 8 architektur umělých neuronových sítí (4 plně konvoluční a 4 konvolučně-rekurentní) a všechny architektury byly za pomoci datasetu natrénovány. Všechna trénování byla provedena desetkrát a následně byla statisticky zpracována pro zvýšení celkové výpovědní hodnoty výsledků. Proces trénování byl následně stručně shrnut a jednotlivé architektury byly porovnány z hlediska času nutného pro dokončení jejich trénování na jednotném HW.

Dalším bodem praktické části bylo podrobení natrénovaných architektur testování na neznámých datech. Stejně jako u trénování byly výsledky každé architektury zpracovány statisticky. Hlavním měřítkem kvality se stala hodnota ROC-AUC (plocha pod ROC křivkou) a na základě těchto hodnot byla jako nejúspěšnější vyhodnocena *čtyřvrstvá konvolučně-rekurentní architektura* (CRNN4) s výsledkem ROC-AUC = $\mathbf{0{,}9046 \pm 0{,}0016}$. Nejlepší architektura byla následně podrobena rozboru ROC křivek jednotlivých tagů a byly zde identifikovány a diskutovány určité problémy, které u některých tagů nastaly.

Celý proces testování byl následně shrnut a dosažené výsledky byly porovnány s výsledky světových prací zabývajících se stejnou nebo podobnou problematikou. Výsledky této práce se ve většině vyrovnaly světovým pracím a v určitých ohledech byly tyto výsledky posunuty dál zejména z hlediska podstatné redukce celkového počtu parametrů neuronových sítí při zachování podobných výsledků.

Jako další krok praktické části byla zvolena tvorba zcela nového datasetu pro další testování navržených architektur neuronových sítí. Tento dataset byl nazván *Last.fm Dataset 2020* a pro jeho tvorbu bylo využito vazby na API služeb Last.fm a Spotify. Výsledný dataset pak obsahoval **100 tagů** a **122 877 skladeb**. Nejúspěšnější architektury z testování na MagnaTagATune Dataset byly následně natrénovány a otestovány i na tomto datasetu. I zde byly výsledky jednotlivých architektur zpracovány statisticky a jako nejúspěšnější byla vyhodnocena *šestivrstvá plně konvoluční architektura* (FCNN6) s výsledkem ROC-AUC = $\mathbf{0{,}8590 \pm 0{,}0011}$. Následně byla tato architektura podrobena rozboru ROC křivek jednotlivých tagů a bylo zde objeveno mnoho nových poznatků, které nebyly s pomocí MagnaTagATune Dataset odhaleny.

Výsledky architektur neuronových sítí na nově vzniklém datasetu představují prvotní hranici hodnot ROC-AUC, kterých lze na jeho základě dosáhnout. Samotný dataset pak skýtá mnoho možností pro další zpracovávání (sloučení podobných tagů, zvýšení počtu skladeb v datasetu pomocí vazby na API výše zmíněných služeb, omezení počtu tagů, volba jiné formy vstupních dat, tvorba specifických architektur neuronových sítí apod.) a představuje tak výzvu pro budoucí práce na téma automatického tagování hudebních děl.

Na úplný závěr praktické části této práce byla vytvořena jednoduchá aplikace, ve které lze otestovat jednotlivé natrénované architektury neuronových sítí na uživatelem zvoleném zvukovém souboru, pro který jsou predikovány pravděpodobnosti jednotlivých tagů.

Samotná problematika automatického tagování hudebních děl nadále skrývá mnoho možností pro budoucí rozvoj zejména z hlediska současné absence kvalitního datasetu, který by byl sestaven například ve spolupráci s muzikology za účelem co nejvěrnějšího přiřazení jednotlivých skladeb konkrétním tagům.

Literatura

[1] KNEES, P.; SCHEDL, M. *Music Similarity and Retrieval: An Introduction to Audio- and Web-based Strategies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, **36**. DOI: 10.1007/978-3-662-49722-7. ISBN 978-3-662-49722-7. Dostupné z URL: <https://doi.org/10.1007/978-3-662-49722-7>

[2] SEMELA, R. *Doporučování hudebního obsahu založené na technikách získávání hudební informace* [online]. Brno, 2018 [cit. 2020-05-12]. Dostupné z URL: <http://hdl.handle.net/11012/82022>. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Tomáš Kiska.

[3] MIKLÁNEK, Š. *Porovnávání zvukových nahrávek za pomoci parametrů popisující barvu zvuku* [online]. Brno, 2017 [cit. 2020-05-12]. Dostupné z URL: <http://hdl.handle.net/11012/68159>. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Tomáš Kiska.

[4] ZEMÁNKOVÁ, Š. *Rozpoznávání hudebního žánru za pomoci technik Music Information Retrieval* [online]. Brno, 2019 [cit. 2020-05-12]. Dostupné z URL: <http://hdl.handle.net/11012/177557>. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Tomáš Kiska.

[5] KÁRNÍK, R. *Rozpoznávání hudebních nástrojů ze zvukových nahrávek za pomoci technik Music Information Retrieval* [online]. Brno, 2019 [cit. 2020-05-12]. Dostupné z URL: <http://hdl.handle.net/11012/177553>. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Tomáš Kiska.

[6] SLÍVOVÁ, M. *Frekvenčně-časová analýza zvukových signálů* [online]. Ostrava, 2017 [cit. 2020-05-12]. Dostupné z URL: <http://hdl.handle.net/10084/118897>. Bakalářská práce. Vysoká škola báňská - Technická univerzita Ostrava.

[7] SMÉKAL, Z. *Číslicové zpracování signálů*. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2012. ISBN 978-80-214-4639-7.

[8] GABOR, D. Theory of Communication. *Journal of the Institution of Electrical Engineers - Part I: General*. 1947, **94**(73), 58-58. DOI: 10.1049/ji-1.1947.0015. ISSN 2054-0582. Dostupné z URL: <https://doi.org/10.1049/ji-1.1947.0015>

[9] DE BENITO-GORRON, D.; LOZANO-DIEZ, A.; TOLEDANO, D.; GONZALEZ-RODRIGUEZ, J. Exploring convolutional, recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset. *EURASIP Journal on Audio, Speech, and Music Processing* [online]. 2019, (1) [cit. 2020-05-12]. DOI: 10.1186/s13636-019-0152-1. ISSN 1687-4722. Dostupné z URL: <https://doi.org/10.1186/s13636-019-0152-1>

[10] RABINER, L.; SCHAFER, R. *Theory and Applications of Digital Speech Processing*. Upper Saddle River: Pearson Prentice Hall, 2011, 1042 s. ISBN 978-0-13-603428-5.

[11] SMÉKAL, Z. *Zpracování řeči*. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2013. ISBN 978-80-214-4896-4.

[12] MCCULLOCH, W.; PITTS, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics* [online]. 1943, **5**(4), 115-133 [cit. 2020-05-21]. DOI: 10.1007/BF02478259. ISSN 0007-4985. Dostupné z URL: <https://doi.org/10.1007/BF02478259>

[13] HEBB, D. *The Organization of Behavior: A Neuropsychological Theory*. New York: JOHN WILEY & SONS, 1949. Dostupné z URL: <http://s-f-walker.org.uk/pubsebooks/pdfs/The_Organization_of_Behavior-Donald_O._Hebb.pdf>

[14] ROSENBLATT, F. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Washington, D.C.: Spartan Books, 1962. Dostupné z URL: <https://hdl.handle.net/2027/mdp.39015039846566>

[15] KOZUMPLÍK, J.; PROVAZNÍK, I. *Umělá inteligence v medicíně*. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav biomedicínského inženýrství, 2007.

[16] MYŠKA, V. *Rekurentní neuronové sítě pro klasifikaci textů* [online]. Brno, 2018 [cit. 2020-05-12]. Dostupné z URL: <http://hdl.handle.net/11012/80785>. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Lukáš Povoda.

[17] STRATIL, J. *Hluboké neuronové sítě pro rozpoznání tváří ve videu* [online]. Brno, 2017 [cit. 2020-05-12]. Dostupné z URL: <http://hdl.handle.net/11012/69856>. Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav počítačové grafiky a multimédií. Vedoucí práce Michal Hradiš.

[18] HANZLÍK, P. *Metody umělé inteligence v rozpoznávání rostlin*. Praha, 2018. Dostupné z URL: <https://www.pef.czu.cz/dl/67670>. Disertační práce. Česká zemědělská univerzita v Praze. Provozně ekonomická fakulta. Katedra informačního inženýrství. Vedoucí práce Arnošt Veselý.

[19] *Matematická biologie učebnice: Algoritmus zpětného šíření chyby (BP algoritmus)* [online]. [cit. 2020-05-16]. Dostupné z URL: <https://portal.matematickabiologie.cz/res/f/neuronove-site-perceptrony.pdf>

[20] RUMELHART, D.; HINTON, G.; WILLIAMS, R. Learning representations by back-propagating errors. *Nature*. 1986, **1986**(323), 533-536. DOI: 10.1038/323533a0. Dostupné z URL: <https://doi.org/10.1038/323533a0>

[21] IOFFE, S.; SZEGEDY, Ch. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ArXiv.org* [online]. Ithaca: Cornell University Library, arXiv.org, 2015 [cit. 2019-05-12]. Dostupné z URL: <https://arxiv.org/abs/1502.03167>

[22] HOCHREITER, S.; SCHMIDHUBER, J. Long Short-Term Memory. *Neural Computation* [online]. MIT Press, 1997, **9**(8), 1735-1780 [cit. 2020-05-11]. DOI: 10.1162/neco.1997.9.8.1735. ISSN 0899-7667. Dostupné z URL: <https://doi.org/10.1162/neco.1997.9.8.1735>

[23] OLAH, Ch. Understanding LSTM Networks. *Colah's blog* [online]. [cit. 2019-11-15]. Dostupné z URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

[24] CHO, K.; GULCEHRE, C.; BAHDANAU, D.; BOUGARES, F.; SCHWENK, H.; BENGIO, Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *ArXiv.org* [online]. Ithaca: Cornell University Library, arXiv.org, 2014 [cit. 2020-05-12]. Dostupné z URL: <https://arxiv.org/abs/1406.1078>

[25] MURPHY, K. *Machine Learning: A Probabilistic Perspective*. Cambridge: MIT Press, 2012, 1067 s. ISBN 978-0-262-01802-9.

[26] KINGMA, D.; BA, J. Adam: A Method for Stochastic Optimization. *ArXiv.org* [online]. Ithaca: Cornell University Library, arXiv.org, 2014 [cit. 2020-05-12]. Dostupné z URL: <https://arxiv.org/abs/1412.6980>

[27] TAUCHMANOVÁ, M. *ROC křivka - nástroj pro hodnocení klasifikačních algoritmů*. Brno, 2007. Dostupné z URL: <https://is.muni.cz/th/d99tt/>. Bakalářská práce. Masarykova univerzita. Přírodovědecká fakulta. Vedoucí práce Marie Forbelská.

[28] Classification: ROC Curve and AUC. *Machine Learning Crash Course* [online]. [cit. 2019-11-20]. Dostupné z URL: <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

[29] CHOI, K.; FAZEKAS, G.; SANDLER, M. Automatic tagging using deep convolutional neural networks. *ArXiv.org* [online]. Ithaca: Cornell University Library, arXiv.org, 2016 [cit. 2020-05-12]. Dostupné z URL: <https://arxiv.org/abs/1606.00298>

[30] NAYYAR, R.; NAIR, S.; PATIL, O.; PAWAR, R.; LOLAGE, A. Content-based auto-tagging of audios using deep learning. *2017 International Conference on Big Data, IoT and Data Science (BID)* [online]. IEEE, 2017, **2017**, 30-36 [cit. 2020-05-12]. DOI: 10.1109/BID.2017.8336569. ISBN 978-1-5090-6593-6. Dostupné z URL: <https://doi.org/10.1109/BID.2017.8336569>

[31] SONG, G.; WANG, Z.; HAN, F.; DING, S.; IQBAL, M. Music auto-tagging using deep Recurrent Neural Networks. *Neurocomputing* [online]. 2018, **292**, 104-110 [cit. 2020-05-12]. DOI: 10.1016/j.neucom.2018.02.076. ISSN 0925-2312. Dostupné z URL: <https://doi.org/10.1016/j.neucom.2018.02.076>

[32] VAN DER WALT, S.; COLBERT, Ch.; VAROQUAUX, G. The NumPy array: a structure for efficient numerical computation. *Computing in Science and Engineering* [online]. Institute of Electrical and Electronics Engineers, 2011, **13**(2), 22-30 [cit. 2020-05-11]. DOI: 10.1109/MCSE.2011.37. ISSN 1521-9615. Dostupné z URL: <https://doi.org/10.1109/MCSE.2011.37>

[33] MCKINNEY, W. Data Structures for Statistical Computing in Python. In: *Proceedings of the 9th Python in Science Conference*. 2010, 2010, s. 56-61. DOI: 10.25080/Majora-92bf1922-00a. Dostupné z URL: <https://doi.org/10.25080/Majora-92bf1922-00a>

[34] HUNTER, J. Matplotlib: A 2D Graphics Environment. *Computing in Science and Engineering*. 2007, **9**(3), 90-95. DOI: 10.1109/MCSE.2007.55. ISSN 1521-9615. Dostupné z URL: <https://doi.org/10.1109/MCSE.2007.55>

[35] ABADI, M.; BARHAM, P.; CHEN, J.; et al. TensorFlow: A system for large-scale machine learning. *ArXiv.org* [online]. Ithaca: Cornell University Library, arXiv.org, 2016 [cit. 2020-05-11]. Dostupné z URL: <https://arxiv.org/abs/1605.08695>

[36] LAW, E.; AHN, L. Input-agreement: A New Mechanism for Collecting Data Using Human Computation Games. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Boston (Massachuttes, USA): ACM, 2009, s. 1197-1206. DOI: 10.1145/1518701.1518881. ISBN 978-1-60558-246-7. Dostupné z URL: <http://doi.org/10.1145/1518701.1518881>

[37] BERTIN-MAHIEUX, T.; ELLIS, D.; WHITMAN, B.; LAMERE, P. The Million Song Dataset. In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*. Miami: University of Miami, 2011, s. 591-596. ISBN 978-0-615-54865-4. Dostupné z URL: <http://ismir2011.ismir.net/ISMIR2011_complete_proceedings.pdf>

Seznam symbolů, veličin a zkratek

ACC	přesnost – Accuracy
ANN	umělá neuronová síť – Artificial Neural Network
AUC	plocha pod křivkou – Area Under the Curve
API	Application Programming Interface
BPG	zpětné šíření chyby - Back-Propagation of Gradient
CD	kompaktní disk – Compact Disc
CUDA	Compute Unified Device Architecture
CPU	procesor – Central Processing Unit
CRNN	konvolučně-rekurentní neuronová síť – Convolutional Recurrent Neural Network
CSV	čárkou oddělené hodnoty – Comma-Separated Values
DFT	diskrétní Fourierova transformace – Discrete Fourier Transform
ELU	Exponential Linear Unit
FCNN	plně konvoluční neuronová síť – Fully Convolutional Neural Network
FFT	rychlá Fourierova transformace – Fast Fourier Transform
FN	falešně negativní – False Negative
FP	falešné pozitivní – False Positive
FPR	míra falešné pozitivity – False Positive Rate
GMM	Gaussovy smíšené modely – Gaussian Mixture Models
GPU	grafický procesor – Graphics Processing Unit
GRU	Gated Recurrent Unit
GUI	grafické uživatelské rozhraní – Graphical User Interface
HW	technické vybavení – Hardware
***k*-NN**	k-nejbližších sousedů – k-Nearest Neighbors
LSTM	Long Short-Term Memory
MFCC	mel-frekvenční kepstrální koeficienty – Mel-Frequency Cepstral Coefficients
MIR	získávání hudebních informací – Music Information Retrieval
MLNN	vícevrstvá neuronová síť – Multi-Layer Neural Network
mRMR	minimální redundance maximální relevance – minimum Redundancy Maximum Relevance
MTT	MagnaTagATune
NN	neuronová síť – Neural Network
PReLU	Parametric Rectified Linear Unit
ReLU	Rectified Linear Unit
RNN	zpětnovazební neuronová síť – Recurrent Neural Network
ROC	operační charakteristika přijímače – Receiver Operating Characteristic
SF(B)FS	Sequential Forward (Backward) Floating Selection
STFT	krátkodobá Fourierova transformace – Short-Time Fourier Transform
SVM	metoda pospůrných vektorů – Support Vector Machines
SW	programové vybavení – Software

TN	pravdivě negativní – True Negative
TP	pravdivě pozitivní – True Positive
TPR	míra pravdivé pozitivity – True Positive Rate

Seznam příloh

A Architektury navržených neuronových sítí

V této příloze jsou podrobně rozepsány architektury navržených neuronových sítí z kapitoly 5.4.

Legenda k následujícím tabulkám:

- `InputLayer` – vstupní vrstva,
- `Reshape` – změna rozměru dat,
- `ZeroPadding2D` – rozšíření okrajů 2D pole o hodnoty 0,
- `Conv2D` – 2D konvoluční vrstva,
- `BatchNormalization` – normalizační vrstva Batch Normalization,
- `ELU` – aktivační funkce typu ELU,
- `MaxPooling2D` – pooling vrstva typu 2D max-pooling,
- `Dropout` – dropout vrstva,
- `GRU` – GRU vrstva,
- `Dense` – plně propojená vrstva.

A.1 FCNN3 (plně konvoluční neuronová síť, 3 vrstvy)

Tab. A.1: Podrobně rozepsaná architektura FCNN3.

Vrstva	**Parametry**	**Výstupní rozměr**
Reshape	target_shape=(96,1366,1)	(None, 96, 1366, 1)
ZeroPadding2D	padding=(0,37)	(None, 96, 1440, 1)
BatchNormalization	axis=3	(None, 96, 1440, 1)
Conv2D	filters=32, kernel_size=(3,3), padding='same'	(None, 96, 1440, 32)
BatchNormalization	axis=3	(None, 96, 1440, 32)
ELU	-	(None, 96, 1440, 32)
MaxPooling2D	pool_size=(4,8)	(None, 24, 180, 32)
Dropout	rate=0.25	(None, 24, 180, 32)
Conv2D	filters=64, kernel_size=(3,3), padding='same'	(None, 24, 180, 64)
BatchNormalization	axis=3	(None, 24, 180, 64)
ELU	-	(None, 24, 180, 64)
MaxPooling2D	pool_size=(4,10)	(None, 6, 18, 64)
Dropout	rate=0.25	(None, 6, 18, 64)
Conv2D	filters=128, kernel_size=(3,3), padding='same'	(None, 6, 18, 128)
BatchNormalization	axis=3	(None, 6, 18, 128)
ELU	-	(None, 6, 18, 128)
MaxPooling2D	pool_size=(6,18)	(None, 1, 1, 128)
Dropout	rate=0.25	(None, 1, 1, 128)
Flatten	-	(None, 128)
BatchNormalization	axis=1	(None, 128)
Dropout	rate=0.5	(None, 128)
Dense	units=50, activation='sigmoid'	(None, 50)

A.2 FCNN4 (plně konvoluční neuronová síť, 4 vrstvy)

Tab. A.2: Podrobně rozepsaná architektura FCNN4.

Vrstva	Parametry	Výstupní rozměr
Reshape	target_shape=(96,1366,1)	(None, 96, 1366, 1)
ZeroPadding2D	padding=(0,37)	(None, 96, 1440, 1)
BatchNormalization	axis=3	(None, 96, 1440, 1)
Conv2D	filters=32, kernel_size=(3,3), padding='same'	(None, 96, 1440, 32)
BatchNormalization	axis=3	(None, 96, 1440, 32)
ELU	-	(None, 96, 1440, 32)
MaxPooling2D	pool_size=(2,5)	(None, 48, 288, 32)
Dropout	rate=0.25	(None, 48, 288, 32)
Conv2D	filters=64, kernel_size=(3,3), padding='same'	(None, 48, 288, 64)
BatchNormalization	axis=3	(None, 48, 288, 64)
ELU	-	(None, 48, 288, 64)
MaxPooling2D	pool_size=(3,6)	(None, 16, 48, 64)
Dropout	rate=0.25	(None, 16, 48, 64)
Conv2D	filters=128, kernel_size=(3,3), padding='same'	(None, 16, 48, 128)
BatchNormalization	axis=3	(None, 16, 48, 128)
ELU	-	(None, 16, 48, 128)
MaxPooling2D	pool_size=(4,6)	(None, 4, 8, 128)
Dropout	rate=0.25	(None, 4, 8, 128)
Conv2D	filters=256, kernel_size=(3,3), padding='same'	(None, 4, 8, 256)
BatchNormalization	axis=3	(None, 4, 8, 256)
ELU	-	(None, 4, 8, 256)
MaxPooling2D	pool_size=(4,8)	(None, 1, 1, 256)
Dropout	rate=0.25	(None, 1, 1, 256)
Flatten	-	(None, 256)
BatchNormalization	axis=1	(None, 256)
Dropout	rate=0.5	(None, 256)
Dense	units=50, activation='sigmoid'	(None, 50)

A.3 FCNN5 (plně konvoluční neuronová síť, 5 vrstev)

Tab. A.3: Podrobně rozepsaná architektura FCNN5.

Vrstva	Parametry	Výstupní rozměr
Reshape	target_shape=(96,1366,1)	(None, 96, 1366, 1)
ZeroPadding2D	padding=(0,37)	(None, 96, 1440, 1)
BatchNormalization	axis=3	(None, 96, 1440, 1)
Conv2D	filters=32, kernel_size=(3,3), padding='same'	(None, 96, 1440, 32)
BatchNormalization	axis=3	(None, 96, 1440, 32)
ELU	-	(None, 96, 1440, 32)
MaxPooling2D	pool_size=(2,2)	(None, 48, 720, 32)
Dropout	rate=0.25	(None, 48, 720, 32)
Conv2D	filters=64, kernel_size=(3,3), padding='same'	(None, 48, 720, 64)
BatchNormalization	axis=3	(None, 48, 720, 64)
ELU	-	(None, 48, 720, 64)
MaxPooling2D	pool_size=(2,4)	(None, 24, 180, 64)
Dropout	rate=0.25	(None, 24, 180, 64)
Conv2D	filters=128, kernel_size=(3,3), padding='same'	(None, 24, 180, 128)
BatchNormalization	axis=3	(None, 24, 180, 128)
ELU	-	(None, 24, 180, 128)
MaxPooling2D	pool_size=(2,5)	(None, 12, 36, 128)
Dropout	rate=0.25	(None, 12, 36, 128)
Conv2D	filters=256, kernel_size=(3,3), padding='same'	(None, 12, 36, 256)
BatchNormalization	axis=3	(None, 12, 36, 256)
ELU	-	(None, 12, 36, 256)
MaxPooling2D	pool_size=(3,6)	(None, 4, 6, 256)
Dropout	rate=0.25	(None, 4, 6, 256)
Conv2D	filters=512, kernel_size=(3,3), padding='same'	(None, 4, 6, 512)
BatchNormalization	axis=3	(None, 4, 6, 512)
ELU	-	(None, 4, 6, 512)
MaxPooling2D	pool_size=(4,6)	(None, 1, 1, 512)
Dropout	rate=0.25	(None, 1, 1, 512)
Flatten	-	(None, 512)
BatchNormalization	axis=1	(None, 512)
Dropout	rate=0.5	(None, 512)
Dense	units=50, activation='sigmoid'	(None, 50)

A.4 FCNN6 (plně konvoluční neuronová síť, 6 vrstev)

Tab. A.4: Podrobně rozepsaná architektura FCNN6.

Vrstva	Parametry	Výstupní rozměr
Reshape	target_shape=(96,1366,1)	(None, 96, 1366, 1)
ZeroPadding2D	padding=(0,37)	(None, 96, 1440, 1)
BatchNormalization	axis=3	(None, 96, 1440, 1)
Conv2D	filters=32, kernel_size=(3,3), padding='same'	(None, 96, 1440, 32)
BatchNormalization	axis=3	(None, 96, 1440, 32)
ELU	-	(None, 96, 1440, 32)
MaxPooling2D	pool_size=(2,2)	(None, 48, 720, 32)
Dropout	rate=0.25	(None, 48, 720, 32)
Conv2D	filters=64, kernel_size=(3,3), padding='same'	(None, 48, 720, 64)
BatchNormalization	axis=3	(None, 48, 720, 64)
ELU	-	(None, 48, 720, 64)
MaxPooling2D	pool_size=(2,2)	(None, 24, 360, 64)
Dropout	rate=0.25	(None, 24, 360, 64)
Conv2D	filters=128, kernel_size=(3,3), padding='same'	(None, 24, 360, 128)
BatchNormalization	axis=3	(None, 24, 360, 128)
ELU	-	(None, 24, 360, 128)
MaxPooling2D	pool_size=(2,3)	(None, 12, 120, 128)
Dropout	rate=0.25	(None, 12, 120, 128)
Conv2D	filters=256, kernel_size=(3,3), padding='same'	(None, 12, 120, 256)
BatchNormalization	axis=3	(None, 12, 120, 256)
ELU	-	(None, 12, 120, 256)
MaxPooling2D	pool_size=(2,4)	(None, 6, 30, 256)
Dropout	rate=0.25	(None, 6, 30, 256)
Conv2D	filters=512, kernel_size=(3,3), padding='same'	(None, 6, 30, 512)
BatchNormalization	axis=3	(None, 6, 30, 512)
ELU	-	(None, 6, 30, 512)
MaxPooling2D	pool_size=(2,5)	(None, 3, 6, 512)
Dropout	rate=0.25	(None, 3, 6, 512)
Conv2D	filters=1024, kernel_size=(3,3), padding='same'	(None, 3, 6, 1024)
BatchNormalization	axis=3	(None, 3, 6, 1024)
ELU	-	(None, 3, 6, 1024)
MaxPooling2D	pool_size=(3,6)	(None, 1, 1, 1024)
Dropout	rate=0.25	(None, 1, 1, 1024)
Flatten	-	(None, 1024)
BatchNormalization	axis=1	(None, 1024)
Dropout	rate=0.5	(None, 1024)
Dense	units=50, activation='sigmoid'	(None, 50)

A.5 CRNN3 (konvolučně-rekurentní neuronová síť, 3 vrstvy)

Tab. A.5: Podrobně rozepsaná architektura CRNN3.

Vrstva	Parametry	Výstupní rozměr
Reshape	target_shape=(96,1366,1)	(None, 96, 1366, 1)
ZeroPadding2D	padding=(0,37)	(None, 96, 1440, 1)
BatchNormalization	axis=3	(None, 96, 1440, 1)
Conv2D	filters=32, kernel_size=(3,3), padding='same'	(None, 96, 1440, 32)
BatchNormalization	axis=3	(None, 96, 1440, 32)
ELU	-	(None, 96, 1440, 32)
MaxPooling2D	pool_size=(4,8)	(None, 24, 180, 32)
Dropout	rate=0.25	(None, 24, 180, 32)
Conv2D	filters=64, kernel_size=(3,3), padding='same'	(None, 24, 180, 64)
BatchNormalization	axis=3	(None, 24, 180, 64)
ELU	-	(None, 24, 180, 64)
MaxPooling2D	pool_size=(4,10)	(None, 6, 18, 64)
Dropout	rate=0.25	(None, 6, 18, 64)
Conv2D	filters=128, kernel_size=(3,3), padding='same'	(None, 6, 18, 128)
BatchNormalization	axis=3	(None, 6, 18, 128)
ELU	-	(None, 6, 18, 128)
MaxPooling2D	pool_size=(6,18)	(None, 1, 1, 128)
Dropout	rate=0.25	(None, 1, 1, 128)
Reshape	target_shape=(1,128)	(None, 1, 128)
GRU	units=256, return_sequences=True	(None, 1, 256)
GRU	units=256, return_sequences=False	(None, 256)
Dropout	rate=0.5	(None, 256)
Dense	units=50, activation='sigmoid'	(None, 50)

A.6 CRNN4 (konvolučně-rekurentní neuronová síť, 4 vrstvy)

Tab. A.6: Podrobně rozepsaná architektura CRNN4.

Vrstva	Parametry	Výstupní rozměr
Reshape	target_shape=(96,1366,1)	(None, 96, 1366, 1)
ZeroPadding2D	padding=(0,37)	(None, 96, 1440, 1)
BatchNormalization	axis=3	(None, 96, 1440, 1)
Conv2D	filters=32, kernel_size=(3,3), padding='same'	(None, 96, 1440, 32)
BatchNormalization	axis=3	(None, 96, 1440, 32)
ELU	-	(None, 96, 1440, 32)
MaxPooling2D	pool_size=(2,5)	(None, 48, 288, 32)
Dropout	rate=0.25	(None, 48, 288, 32)
Conv2D	filters=64, kernel_size=(3,3), padding='same'	(None, 48, 288, 64)
BatchNormalization	axis=3	(None, 48, 288, 64)
ELU	-	(None, 48, 288, 64)
MaxPooling2D	pool_size=(3,6)	(None, 16, 48, 64)
Dropout	rate=0.25	(None, 16, 48, 64)
Conv2D	filters=128, kernel_size=(3,3), padding='same'	(None, 16, 48, 128)
BatchNormalization	axis=3	(None, 16, 48, 128)
ELU	-	(None, 16, 48, 128)
MaxPooling2D	pool_size=(4,6)	(None, 4, 8, 128)
Dropout	rate=0.25	(None, 4, 8, 128)
Conv2D	filters=256, kernel_size=(3,3), padding='same'	(None, 4, 8, 256)
BatchNormalization	axis=3	(None, 4, 8, 256)
ELU	-	(None, 4, 8, 256)
MaxPooling2D	pool_size=(4,8)	(None, 1, 1, 256)
Dropout	rate=0.25	(None, 1, 1, 256)
Reshape	target_shape=(1,128)	(None, 1, 256)
GRU	units=256, return_sequences=True	(None, 1, 256)
GRU	units=256, return_sequences=False	(None, 256)
Dropout	rate=0.5	(None, 256)
Dense	units=50, activation='sigmoid'	(None, 50)

A.7 CRNN5 (konvolučně-rekurentní neuronová síť, 5 vrstev)

Tab. A.7: Podrobně rozepsaná architektura CRNN5.

Vrstva	Parametry	Výstupní rozměr
Reshape	target_shape=(96,1366,1)	(None, 96, 1366, 1)
ZeroPadding2D	padding=(0,37)	(None, 96, 1440, 1)
BatchNormalization	axis=3	(None, 96, 1440, 1)
Conv2D	filters=32, kernel_size=(3,3), padding='same'	(None, 96, 1440, 32)
BatchNormalization	axis=3	(None, 96, 1440, 32)
ELU	-	(None, 96, 1440, 32)
MaxPooling2D	pool_size=(2,2)	(None, 48, 720, 32)
Dropout	rate=0.25	(None, 48, 720, 32)
Conv2D	filters=64, kernel_size=(3,3), padding='same'	(None, 48, 720, 64)
BatchNormalization	axis=3	(None, 48, 720, 64)
ELU	-	(None, 48, 720, 64)
MaxPooling2D	pool_size=(2,4)	(None, 24, 180, 64)
Dropout	rate=0.25	(None, 24, 180, 64)
Conv2D	filters=128, kernel_size=(3,3), padding='same'	(None, 24, 180, 128)
BatchNormalization	axis=3	(None, 24, 180, 128)
ELU	-	(None, 24, 180, 128)
MaxPooling2D	pool_size=(2,5)	(None, 12, 36, 128)
Dropout	rate=0.25	(None, 12, 36, 128)
Conv2D	filters=256, kernel_size=(3,3), padding='same'	(None, 12, 36, 256)
BatchNormalization	axis=3	(None, 12, 36, 256)
ELU	-	(None, 12, 36, 256)
MaxPooling2D	pool_size=(3,6)	(None, 4, 6, 256)
Dropout	rate=0.25	(None, 4, 6, 256)
Conv2D	filters=512, kernel_size=(3,3), padding='same'	(None, 4, 6, 512)
BatchNormalization	axis=3	(None, 4, 6, 512)
ELU	-	(None, 4, 6, 512)
MaxPooling2D	pool_size=(4,6)	(None, 1, 1, 512)
Dropout	rate=0.25	(None, 1, 1, 512)
Reshape	target_shape=(1,128)	(None, 1, 512)
GRU	units=256, return_sequences=True	(None, 1, 256)
GRU	units=256, return_sequences=False	(None, 256)
Dropout	rate=0.5	(None, 256)
Dense	units=50, activation='sigmoid'	(None, 50)

A.8 CRNN6 (konvolučně-rekurentní neuronová síť, 6 vrstev)

Tab. A.8: Podrobně rozepsaná architektura CRNN6.

Vrstva	Parametry	Výstupní rozměr
Reshape	target_shape=(96,1366,1)	(None, 96, 1366, 1)
ZeroPadding2D	padding=(0,37)	(None, 96, 1440, 1)
BatchNormalization	axis=3	(None, 96, 1440, 1)
Conv2D	filters=32, kernel_size=(3,3), padding='same'	(None, 96, 1440, 32)
BatchNormalization	axis=3	(None, 96, 1440, 32)
ELU	-	(None, 96, 1440, 32)
MaxPooling2D	pool_size=(2,2)	(None, 48, 720, 32)
Dropout	rate=0.25	(None, 48, 720, 32)
Conv2D	filters=64, kernel_size=(3,3), padding='same'	(None, 48, 720, 64)
BatchNormalization	axis=3	(None, 48, 720, 64)
ELU	-	(None, 48, 720, 64)
MaxPooling2D	pool_size=(2,2)	(None, 24, 360, 64)
Dropout	rate=0.25	(None, 24, 360, 64)
Conv2D	filters=128, kernel_size=(3,3), padding='same'	(None, 24, 360, 128)
BatchNormalization	axis=3	(None, 24, 360, 128)
ELU	-	(None, 24, 360, 128)
MaxPooling2D	pool_size=(2,3)	(None, 12, 120, 128)
Dropout	rate=0.25	(None, 12, 120, 128)
Conv2D	filters=256, kernel_size=(3,3), padding='same'	(None, 12, 120, 256)
BatchNormalization	axis=3	(None, 12, 120, 256)
ELU	-	(None, 12, 120, 256)
MaxPooling2D	pool_size=(2,4)	(None, 6, 30, 256)
Dropout	rate=0.25	(None, 6, 30, 256)
Conv2D	filters=512, kernel_size=(3,3), padding='same'	(None, 6, 30, 512)
BatchNormalization	axis=3	(None, 6, 30, 512)
ELU	-	(None, 6, 30, 512)
MaxPooling2D	pool_size=(2,5)	(None, 3, 6, 512)
Dropout	rate=0.25	(None, 3, 6, 512)
Conv2D	filters=1024, kernel_size=(3,3), padding='same'	(None, 3, 6, 1024)
BatchNormalization	axis=3	(None, 3, 6, 1024)
ELU	-	(None, 3, 6, 1024)
MaxPooling2D	pool_size=(3,6)	(None, 1, 1, 1024)
Dropout	rate=0.25	(None, 1, 1, 1024)
Reshape	target_shape=(1,128)	(None, 1, 1024)
GRU	units=256, return_sequences=True	(None, 1, 256)
GRU	units=256, return_sequences=False	(None, 256)
Dropout	rate=0.5	(None, 256)
Dense	units=50, activation='sigmoid'	(None, 50)

B Dokumentace přiložených zdrojových souborů

Tato příloha slouží jako základní příručka pro správnou práci se zdrojovými soubory přiloženými k této práci. Funkčnost celého řešení je garantována při použití systému *Windows 10 64-bit (build 1909)* a *Python 3.7.7 64-bit*[1] s nainstalovanými standardními knihovnami doplněnými o knihovny ze souboru `requirements.txt`. Pro správnou funkčnost knihovny *TensorFlow* je zároveň potřeba přítomnost *Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019*[2].

Vhodným způsobem práce je vytvoření a aktivace virtuálního prostředí pro jazyk Python v adresáři se zdrojovými soubory. Toho lze dosáhnout např. pomocí knihovny `virtualenv` a série příkazů v příkazové řádce systému Windows:

```
pip install virtualenv
virtualenv .venv
".venv\Scripts\activate.bat"
```

Potřebné knihovny ke spuštění skriptů lze najít v souboru `requirements.txt` a tyto knihovny lze nainstalovat pomocí jediného příkazu:

```
pip install -r requirements.txt
```

Výše zmíněný postup lze zároveň provést spuštěním skriptu pro příkazovou řádku systému Windows `setup.bat`.

Základní myšlenkou všech skriptů je využívání vytvořené knihovny s názvem `masters`, která obsahuje definice veškerých obecných funkcí, které jsou využívané napříč všemi skripty. Obsah této knihovny není třeba blíže popisovat a lze se s ním seznámit samostatně přímo ve zdrojových kódech.

Za hlavní část lze potom považovat tři spustitelné skripty, které jsou přiblíženy v následujících podkapitolách. Všechny skripty lze konfigurovat pomocí přepínačů přímo z příkazové řádky.

Veškeré zdrojové soubory jsou psány v anglickém jazyce včetně komentářů z důvodu univerzálnosti pro pozdější využití.

B.1 dataset_magnatagatune.py

K sestavení tohoto datasetu je potřeba stáhnout zdrojové soubory datasetu z oficiálního webu. Jedná se o ZIP soubor rozdělený na tři části[3][4][5] obsahující zvuková data. Po rozbalení ZIP souboru je potřeba umístit kompletní stromovou strukturu do adresáře `datasets\magnatagatune_dataset\tracks_mp3`. Některé z MP3 souborů mají příliš dlouhý název, který může v hlubší stromové struktuře systému Windows narazit na

[1] `https://www.python.org/ftp/python/3.7.7/python-3.7.7-amd64.exe`
[2] `https://aka.ms/vs/16/release/vc_redist.x64.exe`
[3] `http://mi.soi.city.ac.uk/datasets/magnatagatune/mp3.zip.001`
[4] `http://mi.soi.city.ac.uk/datasets/magnatagatune/mp3.zip.002`
[5] `http://mi.soi.city.ac.uk/datasets/magnatagatune/mp3.zip.003`

omezení znaků a v takovém případě skript neproběhne korektně. Z tohoto důvodu je doporučeno využít pro spuštění skriptu například adresář `C:\temp`.

Při spuštění skriptu bez jakéhokoli přepínače se spustí všechny kroky sestavení datasetu za sebou. Jednotlivé kroky však lze spustit samostatně pomocí přepínačů, jejichž podrobnější popis lze získat pomocí příkazu:

```
python dataset_magnatagatune.py --help
```

B.2 dataset_lastfm.py

Tento skript slouží k sestavení Last.fm Dataset 2020 podle obr. 7.1. Pro použití tohoto skriptu je potřeba vlastnit funkční API klíč pro Last.fm API[6] a Spotify API[7]. Tyto klíče je nutné zadat přímo do zdrojového kódu skriptu do proměnných `api_key_lastfm` a `api_key_spotify`.

Uvnitř skriptu lze zároveň definovat, pro kolik tagů a pro kolik skladeb na tag dataset sestavit. Jedná se o proměnné `n_tags` a `n_tracks`. Nutno však podotknout, že chování Last.fm API je velmi často nepředvídatelné a některé API dotazy nevrací správná data. Z tohoto důvodu se nastavení těchto parametrů nemusí shodovat s konečnými počty tagů a skladeb ve výsledném datasetu. Ve skriptu zároveň dochází k odstraňování duplicitních skladeb, synonym a nerelevantních tagů a z tohoto důvodu je nastavení těchto proměnných spíše experimentálního rázu. Je doporučeno ponechat tyto proměnné ve výchozím nastavení.

Při spuštění skriptu bez přepínačů jsou postupně provedeny všechny kroky sestavení. Bližší informace k jednotlivým krokům sestavení lze získat pomocí přepínače `--help`:

```
python dataset_lastfm.py --help
```

Sestavený dataset, který byl využit pro trénování a testování navržených architektur v průběhu řešení této práce, lze nalézt v přiložených zdrojových souborech v adresáři `datasets\lastfm_dataset_2020`. Tento dataset již obsahuje napárované vlastnosti ze Spotify API (URL pro stažení MP3 souborů) a z tohoto důvodu není potřeba provádět běh celého skriptu. Data jednotlivých skladeb pro trénování neuronových sítí lze získat pomocí série příkazů:

```
python dataset_lastfm.py --download_spotify_preview
python dataset_lastfm.py --convert_to_wav
python dataset_lastfm.py --compute_melgram
```

[6] `https://www.last.fm/api/`

[7] `https://developer.spotify.com/documentation/web-api/`

B.3 neural_networks.py

Tento skript slouží k natrénování jednotlivých architektur neuronových sítí na sestavených datasetech.

I zde platí, že bližší informace lze získat pomocí přepínače `--help`. V zásadě se však jedná o použití přepínače `--architecture` s určením architektury (`fcnn3`, `crnn3` apod.) a dále pak přepínače `--dataset` s určením datasetu (`magnatagatune` nebo `lastfm`).

Pokud je skript spuštěn bez přepínačů, je spuštěno výchozí nastavení:

```
--architecture crnn4 --dataset magnatagatune
```

B.4 application.py

Skript `application.py` se nachází v samostatném adresáři `application` a obsahuje definici aplikace pro testování architektur neuronových sítí na uživatelem zvoleném zvukovém souboru (viz kapitola 8). Pro vytvoření GUI u této aplikace je využita interní Python knihovna *tkinter*. Aplikace je zamýšlena jako samostatný celek a tudíž je potřeba spustit skript v adresáři `application` pomocí příkazu:

```
python application.py
```

V adresáři `application` lze zároveň nalézt soubor `requirements.txt`, který obsahuje seznam požadovaných Python knihoven pouze pro tuto aplikaci.

C Obsah přiloženého DVD

Přiložené DVD obsahuje veškeré zdrojové soubory, které byly vytvořeny během řešení této práce. Funkčnost všech zdrojových souborů je garantována při použití systému *Windows 10 64-bit (build 1909)* a *Python 3.7.7 64-bit*[1] s nainstalovanými standardními knihovnami doplněnými o knihovny ze souboru `source_files\requirements.txt`. Pro správnou funkčnost knihovny *TensorFlow* je zároveň potřeba přítomnost *Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019*[2].

Podrobnou dokumentaci k jednotlivým skriptům lze nalézt v příloze B. Obdoba celého DVD je zároveň publikována jako GitHub repozitář[3].

```
\ ..... kořenový adresář přiloženého DVD
├── installation_files ..... instalační soubory
│   ├── python-3.7.7-amd64.exe
│   └── VC_redist.x64.exe
├── source_files ..... zdrojové soubory
│   ├── application ..... aplikace pro testování neuronových sítí na zvukových souborech
│   ├── datasets ..... zdrojové soubory datasetů
│   ├── masters ..... vytvořená knihovna funkcí pro Python
│   ├── networks ..... výstupní soubory jednotlivých neuronových sítí
│   ├── dataset_lastfm.py ..... skript pro sestavení Last.fm Dataset 2020
│   ├── dataset_magnatagatune.py ..... skript pro sestavení MagnaTagATune Dataset
│   ├── neural_networks.py ..... skript pro práci s neuronovými sítěmi
│   ├── requirements.txt ..... seznam potřebných Python knihoven
│   └── setup.bat ..... skript pro přípravu virtuálního prostředí Python
└── masters_thesis.pdf ..... text této práce
```

[1] `https://www.python.org/ftp/python/3.7.7/python-3.7.7-amd64.exe`
[2] `https://aka.ms/vs/16/release/vc_redist.x64.exe`
[3] `https://github.com/renesemela/masters-thesis-music-autotagging`