Univerzita Hradec Králové

Fakulta informatiky a managementu

Tagování hudby pomocí strojového učení

Music tagging via machine learning

DIPLOMOVÁ PRÁCE

Autor: Lukáš Rychna

Studijní obor: Aplikovaná Informatika

Vedoucí práce: Ing. Martina Husáková, Ph.D

Hradec Králové 2019

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Jméno a příjmení

Poděkování

Rád bych touto cestou poděkoval Ing. Martině Husákové, Ph.D, za její trpělivost, vstřícný přístup a odborné informace a rady, které mi poskytla během zpracování diplomové práce.

Anotace

Diplomová práce se zabývá vývojem programu pro otagování hudebních dat do žánrů. Hlavním úkolem této práce je vybrat správnou techniku ze současných možných, která dává největší šanci na úspěch.

Práce obsahuje stručný úvod, souhrn metod strojového učení a metody používané pro tagování hudebních dat.

Annotation

This master's thesis focuses on the developement of a program for tagging musical data and genres. The pupose of this study is to choose the best technique, i.e. the one with the highest chances to succed, out of all the techniques available to this date.

The thesis is divided into several parts: brief introduction, overview of the methods of machine learning and suitable methods for musical data tagging.

Klíčová slova

Software, Neuronové sítě, Tensorflow, Keras, Tagování písniček

Keywords

Software, Neural networks, Tensorflow, Keras, Music tagging

Obsah

1 Úvod

V dnešní době je setkávání se s moderními technologiemi prakticky nevyhnutelné. Technologický pokrok proniká do všech odvětví v lidské činnosti a neustále se rozvíjí. Lidé se čím dál více spoléhají na programy a výpočetní techniku a jen těžko se bez ní v dnešní době obejdou.

Primárním cílem předložené diplomové práce je vytvoření softwaru, který dokáže ze vstupních hudebních dat určit, o jaký žánr se jedná. Tento program má smysl pro společnosti, které mají obrovská hudební data v řádech terabitů a musejí ručně označovat (tagovat, oanotovávat) písničky do správných žánrových kategorií.

V současné době chybí na trhu software, který by zvládal s velkou úspěšností hudební data správně otagovat do správných žánrů. Existuje hodně programů, které se zabývají rozpoznáváním řeči. V poslední době, s příchodem algoritmů založených na neuronových sítích, postoupila tato problematika ve vývoji hodně dopředu.

Další skupina programů, která pracuje s hudebními daty, nabízí tagování písní ručně. To při představě s prací v řádech gigabitů je však práce na dlouhou dobu.

Software prezentovaný v diplomové práci je tvořen pro firmu Atteq. Firma Atteq je softwarová společnost, která se zabývá vývojem jak desktopových, tak webových aplikací pro širokou škálu oborů. Například vyvíjí software pro společnost Zoznam.sk, kde vyvíjejí mnoho magazínů. Dále se zabývá správou velké hudební databáze. Jsou tvůrci frameworku ZSL, který slouží pro webové aplikace. V poslední řadě stojí za vývojem aplikace Marblecards, která je svázána s kryptoměnou Etherum.

Diplomová práce je rozdělena do několika částí. V úvodních částech práce se řeší, jaké metody strojového učení existují, a jaké jsou vhodné pro tagování hudebních skladeb. Další část představuje blíže vybranou metodu pro tagování hudebních dat. Dále je popsána samotná implementace metody. Závěrem je předloženo a diskutováno praktické použití softwaru.

2 Cíl práce

Cílem práce je vytvořit počítačový model, který by otagoval písničky podle žánru. K tomu je využito znalostí neuronových sítí a jejich vývoje v informatice. Jako vstup do počítačového modelu jsou brána hudební data bez metainformace žánru písniček. Výstupem modelu je soubor s přiřazeným žánrem pro každou písničku. V teoretické části práce je cílem ukázat, jaké jsou metody strojového učení.

3 Strojové učení a jeho metody

Důležitou vlastností živých organismů je adaptovat se s měnícími se podmínkami prostředí a učit se na základě vlastních zkušeností nebo zkušeností svých předků. Schopnost učit se bývá někdy označována za definici inteligence, proto je snaha touto schopností vybavit stroje. Strojové učení je podoblastí umělé inteligence zabývající se technikami a algoritmy, které dávají strojům schopnost "učit se". V oblasti umělé inteligence máme učením na mysli změnu vnitřního stavu systému, který bude přizpůsobivější vůči změnám okolního prostředí. Strojové učení je velmi propojeno s oblastmi statistiky a získáváním znalostí. Jeho uplatnění se dnes nachází v mnoha oborech, například na finančních trzích, v medicíně a dalších oborech. Strojové učení můžeme rozdělit do dvou základních skupin: učení se znalostem a učení se dovednostem. U prvního typu se hledají koncepty a obecné zákonitosti. U druhého typu se zdokonalují schopnosti při procvičování. Podrobnější dělení strojového učení můžeme rozdělit do následujících kapitol (LE, 2016).

3.1 Rozhodovací stromy

Rozhodovací stromy se používají v řadě oblastí. Při vytváření rozhodovacího stromu se postupuje metodou Rozděl a panuj. Tréninková data jsou na začátku jedna množina a data se postupně rozdělují na menší podmnožiny do té doby, než v dané podmnožině budou data jedné třídy. Cílem metody je nalézt strom konzistentní s tréninkovými daty. Důležité je při této metodě volit správné atributy, podle kterých se budou data dělit na dané podmnožiny. Tomuto postupu se říká „top down induction of decision trees" (TDIDT) algoritmus.

TDIDT algoritmus

1. zvol jeden atribut jako kořen dílčího stromu
2. rozděl data v tomto uzlu na podmnožiny podle hodnot zvoleného atributu a přidej uzel pro každou podmnožinu

3. existuje-li uzel, pro který nepatří všechna data do téže třídy, opakuj pro tento uzel postup od bodu 1, jinak skonči

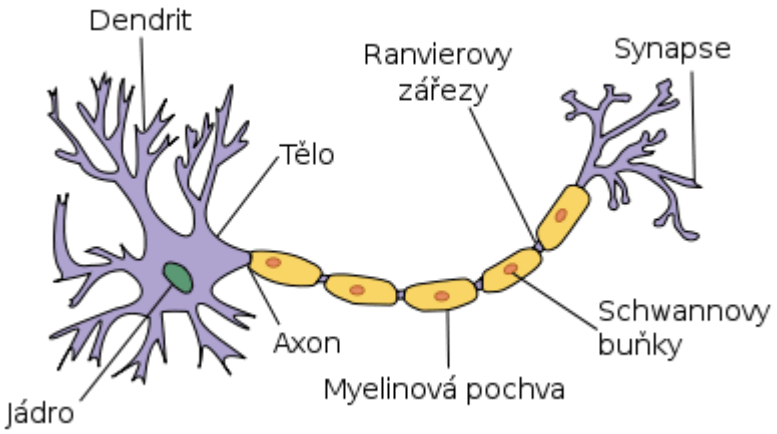
K rozhodnutí, jak vybrat správný atribut dělení se používají nejčastěji tyto charakteristiky atributů: entropie, informační zisk, poměrný informační zisk a Giniho index.

Pro srozumitelnější pochopení znalostí rozhodovacího stromu je možné strom převést na sadu rozhodovacích pravidel.

Rozhodovací stromy se dají dělit na dvě kategorie: klasifikační stromy a regresní stromy. Klasifikační stromy určují dané objekty do daných tříd. Regresní stromy umožňují odhadovat hodnoty numerických atributů. V listových uzlech mají na rozdíl od klasifikačních stromů číselné hodnoty atributů. Rozhodovací stromy jsou vhodné k řešení úloh, kde data jsou reprezentovány hodnotami atributů, cílem je klasifikovat data do konečného počtu tříd a tréninková data můžou obsahovat chyby (BERKA, 2015a).

3.2 Neuronové sítě

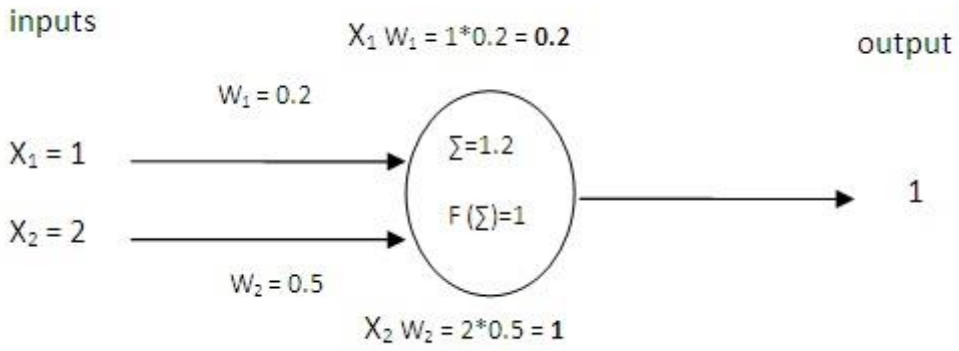
Inspirací pro Neuronové sítě je lidský mozek. Ten je složen z neuronů, kterých je přibližně 10^{11}, jsou propojené navzájem a tvoří jednu síť. Když člověk vidí například psa, jeho vlastnosti jako je barva, tvar, velikost jsou pozorovány očima. Tato data dostane mozek na vstupu a začne je zpracovávat pomocí neuronů. Nejdříve neurony vyhledávají další obrázky psů, které člověk předtím viděl, a porovnávají tyto obrázky psa v paměti člověka s právě viděným psem. Na základě těchto porovnání jsme schopni odpovědět, jakého psa vidíme. Na obrázku 1 lze vidět, jak vypadá lidský neuron a z jakých částí se skládá.



Obrázek 1 – Stavba neuronu (MEDELOVÁ, 2015)

Neuron se skládá z částí těla, jádra, axonů, dendritů, Ranvierových zářezů, Myelinové pochvy, Schwannových buněk a synapsí.

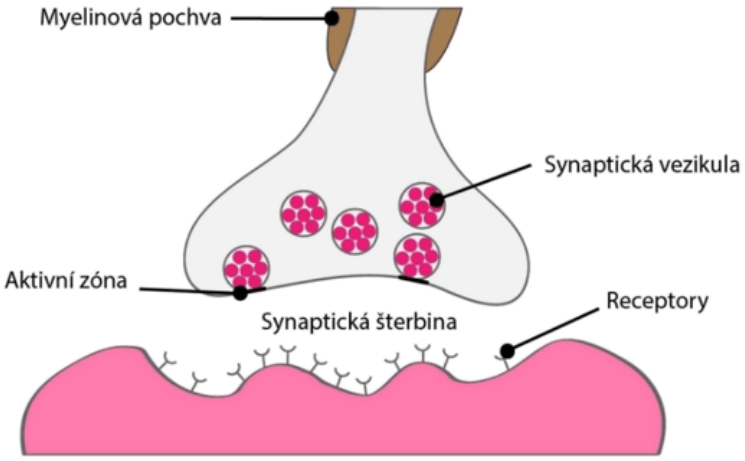
V neuronových sítích se objevují různé typy neuronů. Nejpoužívanějším typem v matematických modelech je neuron používající názvosloví perceptron, který je lineární. V dalších kapitolách budou ukázaný i ostatní typy neuronů. Perceptron byl vynalezen mezi lety 1950 a 1960 vědcem Frankem Rosenblattem, který se inspiroval dřívější prací vědců Warrena McCullocha a Waltera Pittse. Na obrázku 2. je vidět jednoduchý perceptron, který má dva vstupy. Každý vstup má nějakou váhu. Dále existuje v perceptronu jednotka, která sumarizuje váhu * hodnotu pro daný vstup. Na součet hodnot se použije aktivační funkce, která dá konečný výstup.



Obrázek 2 - Jednoduchý perceptron (HASSANKASHI, 2019)

Váha určuje sílu vzájemného propojení mezi dvěma perceptrony. Váha může být pozitivní, což znamená, že přenos dat mezi perceptrony je častý. Čím vyšší je váha, tím je přenos častější. Negativní váha znamená, že mezi perceptrony není častý přenos dat. V lidském mozku váhu značí skladba těchto vlastností: počet dendritů v neuronu, počet synapsí mezi dendrity, pre- a post-synaptické terminály, tvar synaptické mezery, intenzita signálu a významnou částí je myelinová pochva.

Aktivační funkce perceptronu modeluje jeho výstupní signál. V lidském mozku se o vysílání signálu starají chemické látky a elektrický proud. Podrobněji se odehrává to, že se v axonu sčítají informace, které přišly z různých dendritů. Pokud sčítání překročí práh významnosti, začne se signál šířit pomocí elektrického proudu. Následně doputuje do ukončení neuronu a způsobí synapsi. Synapse je proces, který způsobí přenos z jednoho neuronu na druhý. Na konci neuronu jsou váčky plné neurotransmiterů (synaptické vezikuly) a na začátku druhého neuronu jsou receptory. Mezi neurony je mezera a příchodem aktivačního signálu je způsobeno, že obsah synaptických vezikul se přemístí do synaptické štěrbiny. Neurotransmitery slouží jako klíče, a když zapadnou do receptorů signál se šíří dál.



Obrázek 3 – Synapse (MEDELOVÁ, 2015)

Aktivační funkce pro neuronovou síť by neměla být nelineární funkce, například exponenciální. Aktivační funkci se také říká v počítačových sítích přenosová funkce. Nejčastěji se v neuronových sítích používají tyto aktivační funkce (SHARMA, 2017):

1. Lineární funkce(Identická)

   Lineární funkce není vhodná pro neuronové sítě. Do neuronových sítí je snaha dostat nelinearitu. Předpis identické funkce je $f(x) = x$.

   Její rozsah není omezen.

2. Sigmoid funkce

   Hlavní důvod užití Sigmoid funkce, je že se zobrazuje v rozsahu (0,1). Předpis funkce je $f(x) = \frac{1}{1+e^{-x}}$. Často se používá v modelech, kde jako výstup je pravděpodobnost daného jevu.

3. Softmax funkce

   Je zobecnění funkce Sigmoid, kde dostáváme výsledné pravděpodobnosti pro více tříd. Předpis funkce je $f(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{n} x_j}\ pro\ i = 0, .., n$. Výsledné hodnoty pro každou třídu jsou v rozsahu (0,1] a jejich součet je dohromady vždy 1. Hodně se využívá při klasifikaci v neuronových sítích v poslední vrstvě (POLAMURI, 2017).

4. Tanh funkce

   Je podobná funkci Sigmoid, ale většinou s ní je dosahováno lepších výsledků. Předpis funkce je $f(x) = tanh(x)$. Hodnoty funkce se zobrazují v rozsahu (-1,1). To je výhoda protože negativní vstupy budou zobrazovány negativně. Využívá se v dopředných neuronových sítích.

5. Relu funkce

   Populární aktivační funkce v posledním období. K jejímu rozmachu pomohl vývoj Konvolučních sítí a hlubokého učení. Předpis funkce je $f(x) = max(0, x)$. Relu funkce záporné hodnoty nastavuje na nulu, což někdy může snižovat kvalitu modelu.

6. Leaky Relu funkce

   Má předpis funkce takový, že pro kladné hodnoty je funkce Identická a pro záporné je definována jako $f(x) = a * x$, kde a je konstanta, která má malou hodnotu. Většinou bývá voleno 0.01.

7. Jednotkový skok

Má předpis takový, že pro všechny záporné hodnoty je zobrazena nula. Pro kladná čísla je zobrazena jednička.

Bias

Bias je konstanta, která pomáhá s aktivací aktivační funkce. Biasova konstanta umožňuje modelu měnit aktivační funkci skrz tuto konstantu. Pomáhá najít nejlepší nastavení modelu pro daná data (SHEKHAR, 2018).

V některých typech neuronových sítí se kromě dopředených funkcí objevují i funkce zpětné propagace, které pomáhají k dosažení lepších výsledků a používají se pro učení modelu.

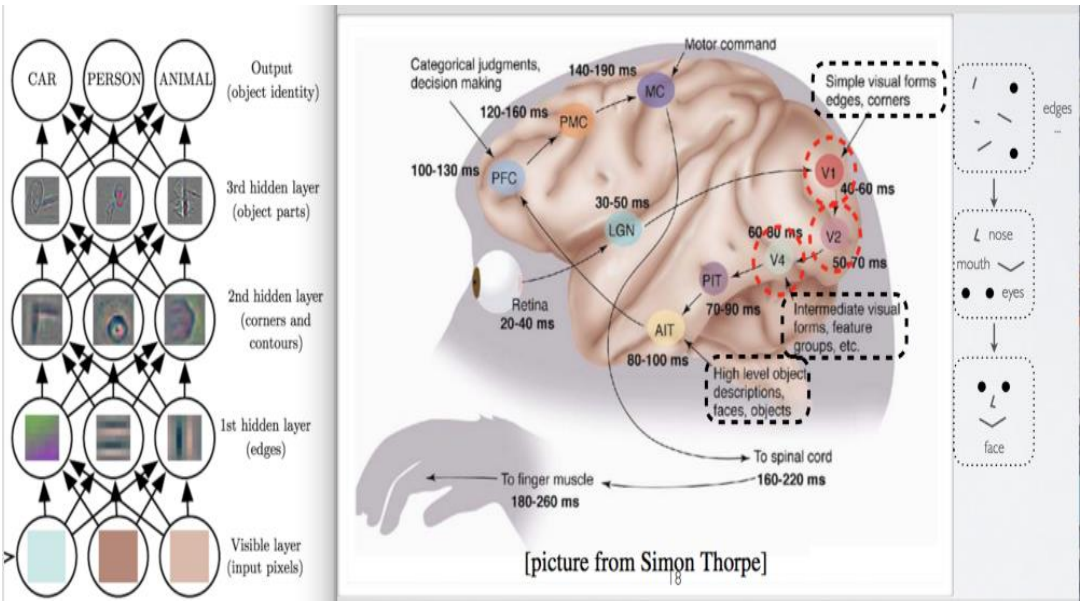
Funkce zpětné propagace

Funkce zpětné propagace je algoritmus, který přepočítává váhy mezi neurony, jimiž prošel výstupní signál, což nám umožňuje velice efektivně zlepšovat neuronovou síť. Zpětná propagace je o pochopení, jak změny vah mezi neurony a Biasovi konstanty mění výsledky aktivační funkce pro daný neuron. V důsledku to znamená výpočet parciálních derivací a úpravu vah v síti (GUPTA, 2017a).

Neuronové sítě se dále dělí na další podskupiny. Zde nebudou uvedeny všechny možné typy, ale jen zástupci nejpoužívanějších typů. Existují dále typy neuronových sítí, které mezi sebou kříží tyto základní typy (MALADKAR, 2018).

3.2.1 Dopředné neuronové sítě

Inspirací pro dopředné sítě je také lidský mozek. Na následujícím obrázku 5 je znázorněno, jak pracuje mozek při vyhodnocení, resp. co vidíme za objekt.



Obrázek 4 – Dopředná síť, jak je utvořena v mozku (GUPTA, 2017b)

V levé části obrázku 4. vidíme, jak mozek rozpoznává objekty pomocí různých vrstev. Každá vrstva charakterizuje objekt podle odlišných vlastností. Například v první vrstvě se rozpoznává podle hran objektů, v druhé vrstvě podle kontury a rohů, třetí vrstva rozpoznává podle tvaru částí objektu. Na pravé části obrázku vidíme, jak a kde probíhá vyhodnocení očního smyslu.

Poskládáním perceptronů do více vrstev vznikají sítě známé jako Dopředné neuronové sítě. Cílem těchto sítí je aproximovat obecné funkce f*, které pro vstup x přiřazují kategorii y. $y = f^*(x)$. Dopředné neuronové sítě definujeme jako $y = f(x; \Theta)$, kde Θ je parametr z učení, který dává nejlepší aproximaci funkce f^*. Modely těchto sítí se nazývají Dopředné, protože neexistuje žádná zpětná vazba mezi vrstvami, kdy by byla výstupní data předána zpátky do výpočtu modelu. Sítě, které toto umožňují, se nazývají Rekurentní, o nich bude psáno v kapitole 3.2.4.

Máme-li Dopřednou neuronovou síť postavenou z perceptronů a chceme, aby se naučila vyřešit daný problém, musíme ji nejdříve vytrénovat. Perceptronová síť nám to umožnuje svojí vlastností, že když se jenom trochu změní vstup, tak se jenom trochu změní výstup.

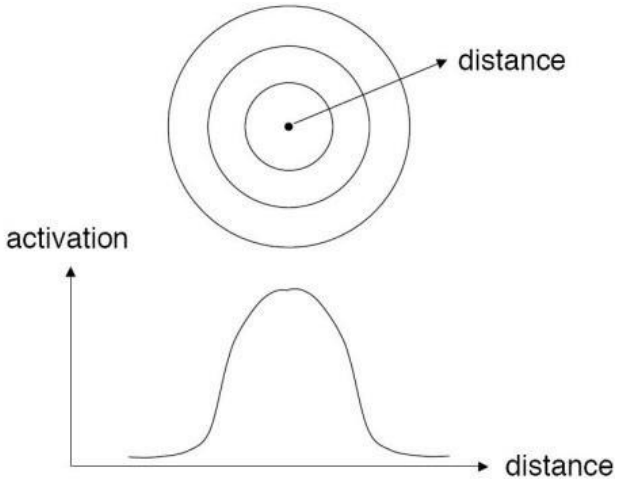
Architektura sítě je taková, že první vrstvě se říká vstupní a jejím neuronům (perceptronům) vstupní neurony. Poslední vrstva je známa jako výstupní vrstva, která obsahuje výstupní neurony. Výstupní neurony reprezentují výsledek dané úlohy. Vrstvám, které jsou mezi první a poslední vrstvou, se říká skryté vrstvy, jejich neurony propojují vstupní a výstupní neurony. Universální theorém o aproximaci říká: Pokud máme Dopřednou síť s lineární vstupní vrstvou, aspoň jednou skrytou vrstvou, jakoukoliv aktivační funkcí a jakoukoliv Borelovskou funkci, můžeme zobrazovat vstup jednoho prostoru do výstupu druhého s jakýmkoliv požadovaným nenulovým množstvím chyby za předpokladu, že je v síti dostatečné množství skrytých neuronů. Jednoduše řečeno existuje dostatečně velká síť pro daný problém, aby dosáhla jakékoliv míry přesnosti, kterou chceme. Neříká však, jak má být síť velká.

Chybová funkce pomáhá při učení modelu. Chybová funkce nepočítá, kolik správných výsledků dostaneme po použití našeho modelu, ale kolik špatných. Problém s počítáním dobrých výsledků je ten, že když změníme trochu model, většinou dostaneme správných výsledků stejně. Proto je těžké určit, jak změnit váhy a propojení neuronů ke zlepšení modelu. Chybovou funkci nejčastěji představuje nějaká kvadratická funkce. Například funkce metody nejmenších čtverců. $C(w,b) \equiv \frac{1}{2n}\sum_x \|f(x) - a\|^2$, kde w značí váhu všech uzlů, b jsou biasové hodnoty všech uzlů, n je množství trénovacích vstupů, a je vektor výstupů pro každý daný vstup x. Chyba samozřejmě záleží i na w a b, ale pro zjednodušení je vzorec zkrácen. C nazýváme jako střední kvadratickou chybu (MSE).

Navrhování a trénování neuronové sítě se příliš neliší od trénování ostatních strojových učení s gradientním sestupem. Největším rozdílem mezi neuronovými modely a lineárními modely je nelinearita neuronových sítí, která způsobuje, že nejzajímavější chybové funkce se stávají nekonvexní. Z toho vyplývá, že neuronové sítě jsou nejčastěji vycvičovány pomocí iterativních optimalizátorů založených na gradientech, které pouze řídí chybovou funkci na velmi nízkou hodnotu (GUPTA, 2017b).

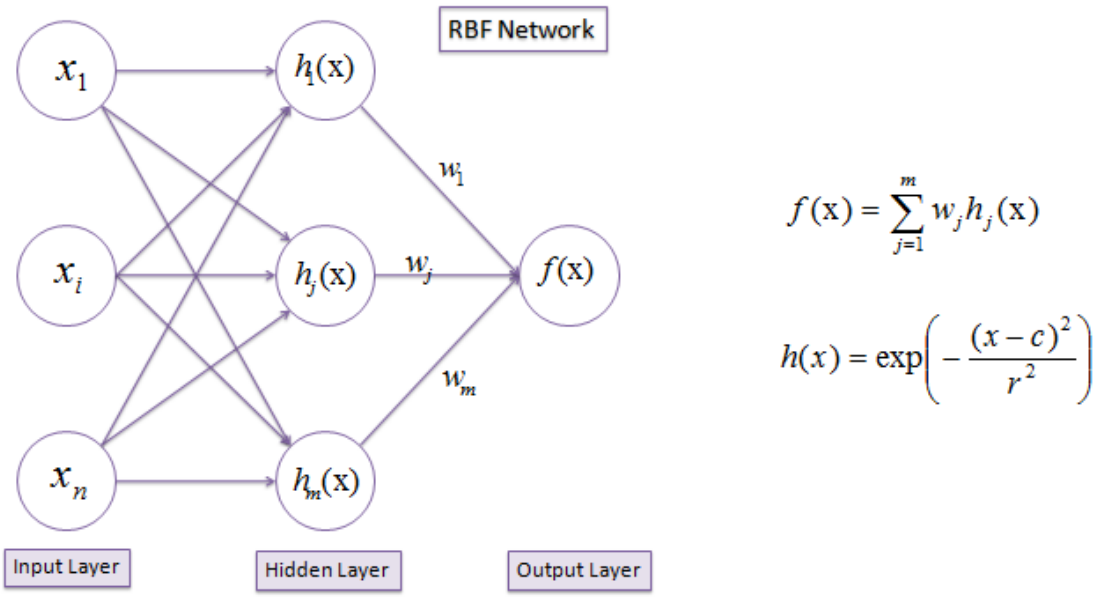
3.2.2 Neuronové sítě s radiální bází

Neuronové sítě s radiální bází (regression-based neural network - RBNN) mívají typicky tři vrstvy: vstupní, výstupní a jednu skrytou vrstvu. Výpočet se provádí pouze ve skryté a výstupní vrstvě. V síti se objevují dva typy neuronů. První typ je perceptron, nám známý už z Dopředných sítí. Druhý typ je radiální neuron, který jako klasifikační funkci využívá radiální funkci. Definuje se receptor t, ke kterému se potom počítá vzdálenost vstupních bodů. Nejčastěji se jako metrika používá Gausova funkce. Radiální vzdálenost tedy definujeme jako $r = \|x - t\|$, radiální aktivační funkce je potom definována následovně $\emptyset(r) = \exp(-r^2/2s^2)$, kde s > 0.



Obrázek 5 – Radiální vzdálenost a radiální funkce s konfrontační mapou (CHANDRADEVAN, 2017)

Radiální neurony se používají ve skryté vrstvě. Perceptrony se používají ve výstupní vrstvě. Na následujícím obrázku 6. je vidět, jak taková neuronová síť s radiální bází vypadá.



Obrázek 6 – Neuronová síť s radiálními neurony (CHANDRADEVAN, 2017)

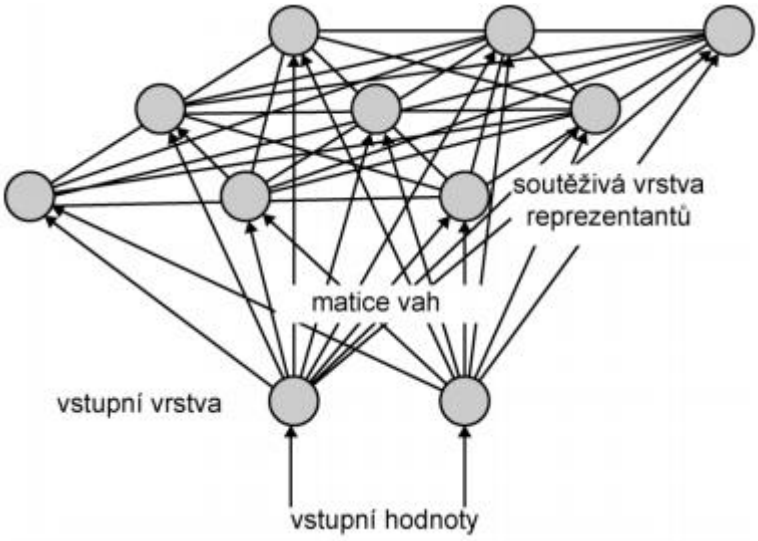
Učení sítě probíhá následovně. Nejdříve se optimalizuje učení skryté vrstvy pomocí zpětné propagace a shlukové analýzy. Pro každý radiální neuron ve skryté vrstvě musíme najít receptory a jejich rozptyl šíření radiální bázové funkce. Receptory se hledají pomocí shlukové analýzy. Pro každý receptor můžeme najít rozptyl pomocí metody čtvercových součtů vzdáleností mezi receptory. V druhé fázi výcviku se aktualizují váhové vektory mezi skrytou a výstupní vrstvou (CHANDRADEVAN, 2017).

Učení RBNN je rychlejší než u více vrstevnatých sítí s lineárními neurony (Multi-layer Perceptron - MLP). Ze sítě je jednoznačněji vidět, co který uzel dělá. Klasifikace trvá naopak déle oproti MLP. RBNN se používají pro klasifikační a regresní úlohy.

3.2.3 Kohonenova samoorganizační neuronová síť

Kohonenova samoorganizační neuronová síť slouží k řešení problémů, kde je třeba rozhodnout, rozlišit nebo roztřídit dané objekty. A to díky vlastnostem samoorganizace a shlukování objektů s podobnými vlastnostmi do stejných tříd. Struktura Kohonenovy samoorganizační neuronové sítě je tvořena dvěma vrstvami.

První vrstva je vstupní a druhé se říká kompetiční. Tyto vrstvy jsou plně propojeny. Kompetiční vrstva má neurony uspořádané do geometrické struktury, nejběžněji to bývá přímka nebo mřížka, občas se vyskytuje hexagonální tvar. Neurony nacházející se v kompetiční vrstvě se nazývají také detektory. Geometrické uspořádání detektorů umožňuje definovat okolí každého detektoru (INSTITUT BIOSTATISTIKY A ANALÝZ MASARYKOVY UNIVERZITY, 2018).



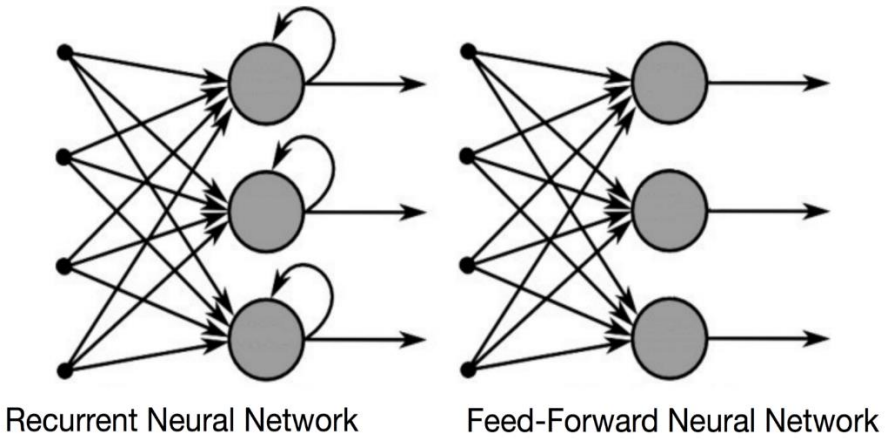
Obrázek 7 - Samoorganizující se neuronová síť (INSTITUT BIOSTATISTIKY A ANALÝZ MASARYKOVY UNIVERZITY, 2018)

Učení probíhá na základě předkládání vstupů z trénovací množiny. Na začátku jsou náhodně nastaveny váhy mezi vstupní a kompeteční vrstvou. Dále se postupně překládají vstupní hodnoty ze vstupní množiny $I = \{x^1, x^2, .., x^n\}$. V každém kroku vyhrává jeden detektor na základě porovnání Euklidovské vzdálenosti a vybrání detektoru s nejmenší Euklidovskou vzdáleností vůči vstupnímu vektoru x^i a tomu je přepočítána váha na základě vzorce $w_i(k+1) = w_i(k) + \mu(x(k) - w_i(k))$, kde μ představuje hodnotu z intervalu (0,1> . Proměnná μ říká, jak se posunuje vektor w směrem k vektoru x, a udává rychlost učení. Změnou vah posouváme vítězný detektor blíže k aktuálnímu vstupu. Takto postupujeme tak dlouho, dokud jsou změny zanedbatelné. Když je neuronová síť vytrénovaná, klasifikace probíhá stejně, jen se nepřepočítávají váhy (VOJÁČEK, 2006).

3.2.4 Rekurentní neuronové sítě

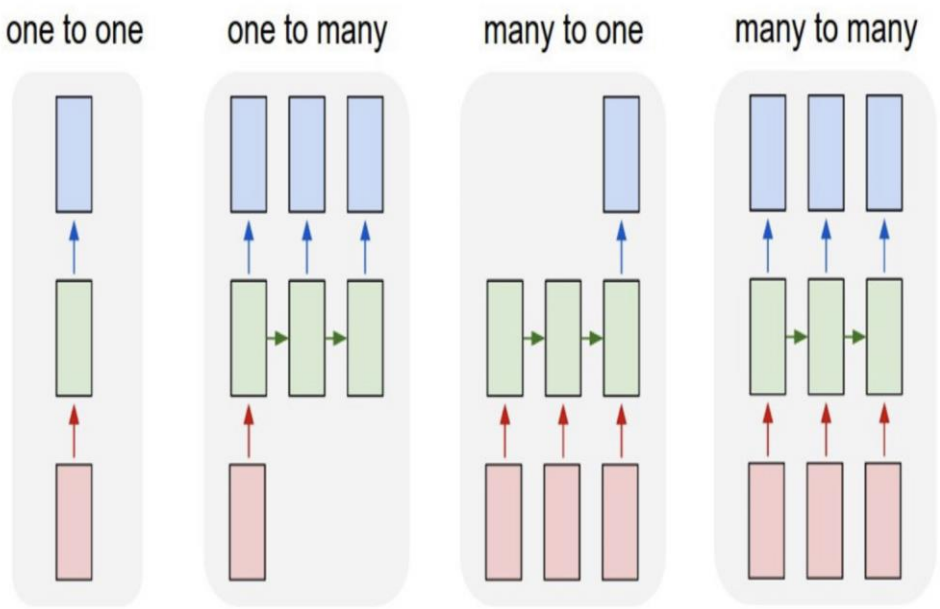
Rekurentní neuronové sítě (Recurrent neural network - RNN) jsou velmi používaným typem neuronových sítí a velkým příslibem v budoucnosti. Jako jedny z mála používají vnitřní paměť. RNN byly vynalezeny v 80. letech minulého století, ale jejich hlavní rozvoj přišel až s pozdější dobou, kdy byla dostupná větší výpočetní kapacita. Vnitřní paměť umožňuje síti předpovídat výsledky a lépe pochopit význam sekvencí a jejich kontext. To je důvod proč mají RNN velké uplatnění napříč obory. Využívají se například ve zpracování řeči, textu, obrázků, zvuků, finančních dat, počasí a v dalších oborech.

Základní RNN užívají k pamatování smyčky. Při rozhodování zohledňují současný vstup a také co se naučily z předešlých vstupů. V tom se liší od dopředných neuronových sítí.



Obrázek 8 – Rozdíl mezi RNN a Dopřednou neuronovou sítí (DONGES, 2018)

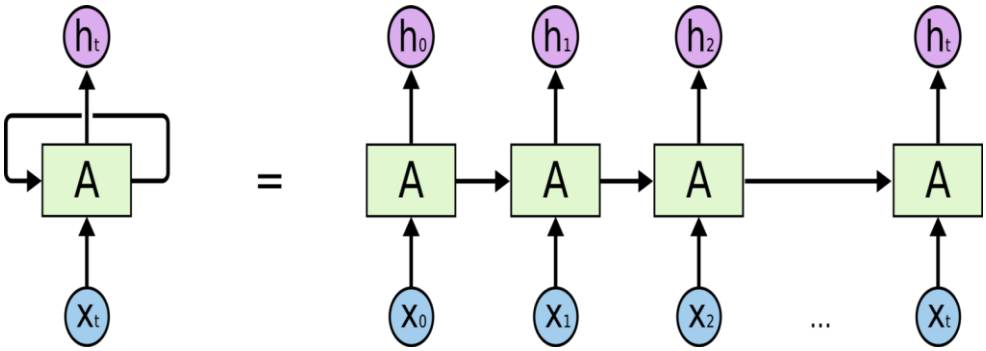
Normální RNN mají krátkodobou paměť v jednotlivých neuronech, objevem buňky Long short-term memory (LSTM) si můžeme pamatovat věci déle. Dalším rozdílem mezi dopřednou neuronovou sítí a RNN je, že dopředné sítě zobrazují jeden vstup na jeden výstup. To rekurentní sítě nabízejí různé možnosti zobrazování, jak je vidět na následujícím obrázku 9.



Obrázek 9 – Možnosti propojení neuronů v RNN (DONGES, 2018)

Funkce zpětné propagace v čase

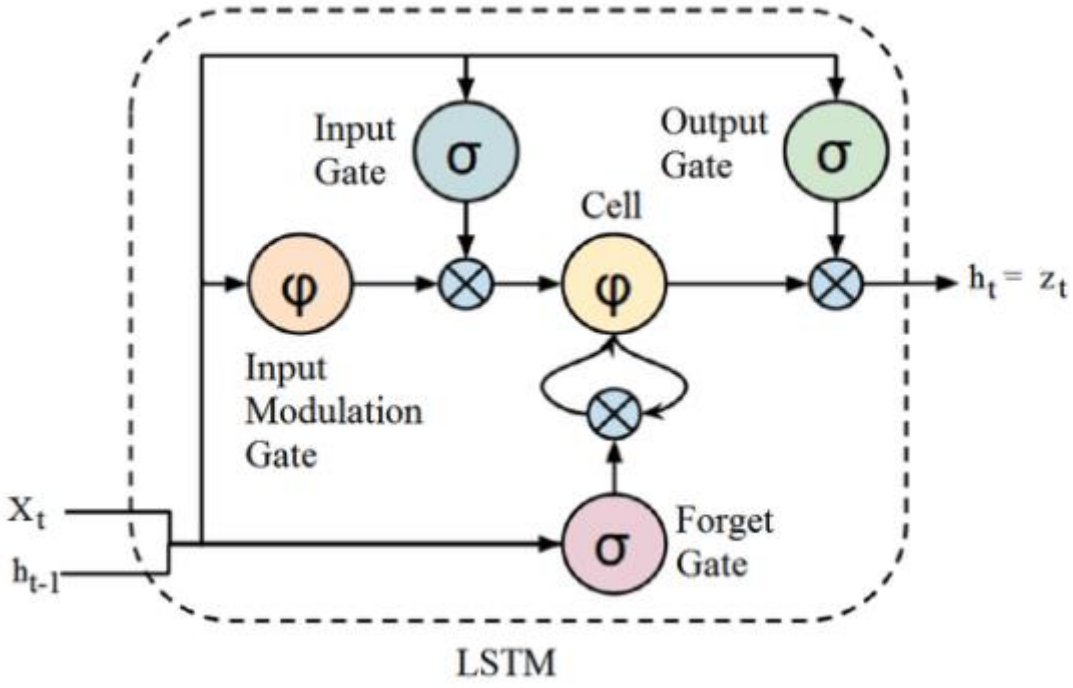
Funkce zpětné propagace v čase probíhá obdobně jako při dopředných sítích, jen průchod zpět je složitější. Můžeme říci, že RNN je sekvence neuronových sítí, kde cvičíme postupně jednu síť za druhou se zpětnou propagací. Na obrázku 10 můžeme vidět znázornění rozložení RNN na několik sítí v čase. V zpětné propagaci v čase je třeba provést rekonstrukci, jak se chyba šířila v čase. Chyba se šíří od poslední časové změny až k začátku, proto můžeme v každém časovém úseku spočítat velikost chyby a aktualizovat váhy v síti. Výpočet může být časově dost náročný, pokud je počet časových úseků velký.



Obrázek 10 – Rozložení RNN v čase (DONGES, 2018)

LSTM

LSTM je rozšíření pro RNN, které přidává sítím dlouhodobější paměť. Jednotky LSTM se používají jako prvky pro stavbu vrstev v RNN. Takové sítě se potom často nazývají LSTM sítě. LSTM se trochu podobá pamětím počítačů. Umí číst, zapisovat a odstraňovat informace z paměti.



Obrázek 11 – LSTM buňka (GANDHI, 2018)

LSTM má čtyři brány do buňky: vstupní, výstupní, zapomínající bránu a modulační vstupní bránu. Tyto brány rozhodují o tom, jak se zachází s informacemi, jestli se

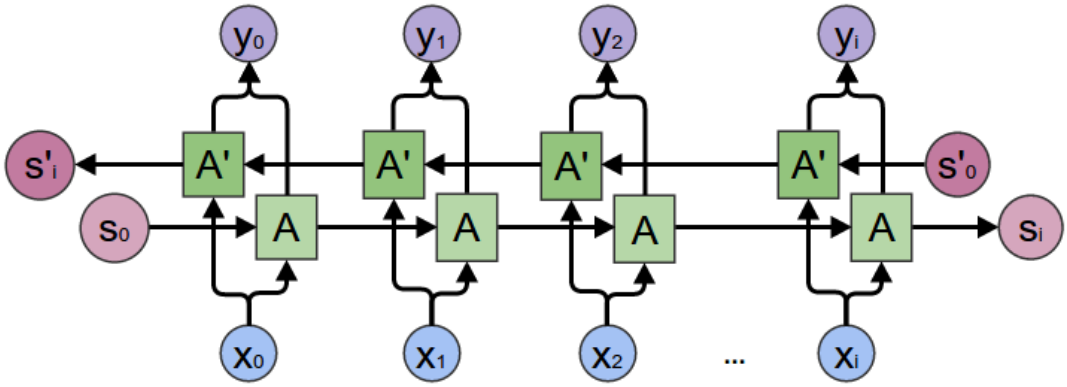
uschovává nový vstup ve vstupní bráně nebo se smaže informace tím, že je poslána do zapomínající brány nebo se pošle v aktuálním kroku informace na výstup. Modulační brána udržuje synchronizaci s ostatními bránami (GANDHI, 2018).

Gated recurrent unit (GRU)

Další buňka, která je základním kamenem pro další typ RNN sítí je GRU. GRU buňka se od LSTM liší tím, že má jenom tři brány: aktualizační, reset a modulační bránu. Aktualizační brána se stará o to, kolik informací z minulosti by mělo být posláno dál. Reset brána rozhoduje o tom, kolik informací z minulosti bude vyřazeno. GRU je výpočetně efektivnější, ale LSTM bývá výkonnostněji většinou lepší (GANDHI, 2018).

Bidirectional RNN (BiRNN)

Někdy se pro rozhodnutí problému nestačí jen učit z minulosti, ale je třeba se učit z budoucnosti. K tomu právě slouží návrh BiRNN, který mezi sebou propojuje neurony z minulosti i z budoucnosti. Jak vypadá schéma BiRNN vidíme na obrázku 12.

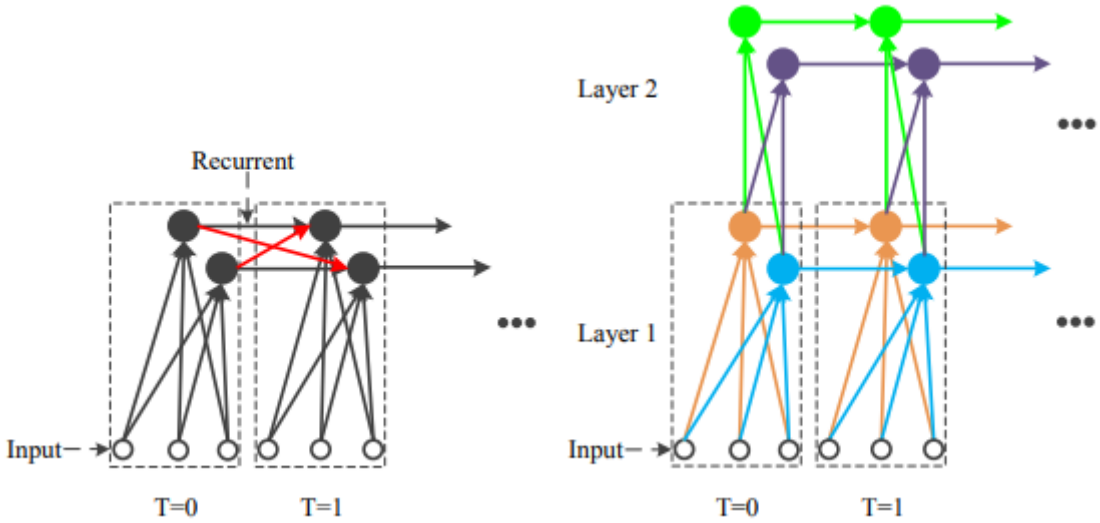


Obrázek 12 – BiRNN schéma (GANDHI, 2018)

Propagace hodnot do vyšších vrstev se provádí ve dvou krocích. Nejdříve začínáme s prvním časovým neuronem a počítáme v neuronech dopředné výsledky (A). Až dojdeme na poslední časový neuron, tak začneme počítat zpětné výsledky (A'). Po skončení zpětných výpočtů dostaneme výsledný výstup z neuronů kombinací A a A'. Spojením BiRNN a LSTM buněk dostáváme výkonnější buňky. Také existují obdobné buňky spojením BiRNN a GRU (GANDHI, 2018).

Independent RNN (IndRNN)

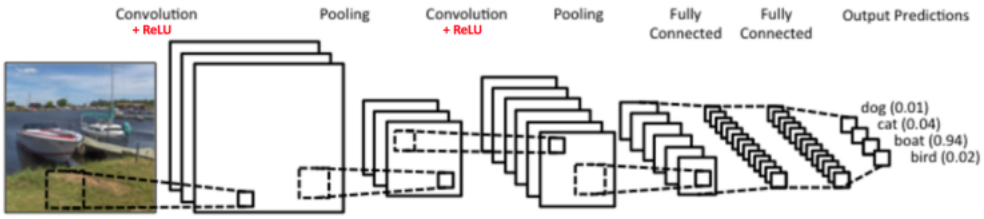
IndRNN neurony nejsou závislé na žádné jiném neuronu ze stejné nebo vyšší vrstvy. To má hlavní výhodu, že při trénování modelu a počítání zpětné propagace se nešíří chyba skrz celou síť. V IndRNN sítích můžeme budovat daleko větší sítě než při LSTM, protože IndRNN má větší schopnost déle si pamatovat, a tím dokáže zpracovat víc kroků výpočtu. Více vrstevnaté IndRNN efektivně zvyšují hloubku sítě. Na obrázku 13 je zobrazen rozdíl struktury mezi normální RNN a IndRNN (LI, LI, COOK, ZHU a GAO, 2018), (LI, 2018).



Obrázek 13 - Rozdíl mezi RNN a IndRNN (LI Shuai, 2018)

3.2.5 Konvoluční neuronové sítě

Konvoluční neuronové sítě (Convolutional neural network - CNN) jsou velmi efektivní v rozpoznávání a klasifikaci obrázků. V praxi se používají například pro identifikaci obličejů, v autonomních autech a v řadě dalších případů. Základní architekturou pro CNN byla LeNet. Byla vyvinuta v roce 1988 a sloužila hlavně k rozpoznávání znaků a číslic. Z architektury LeNet pak vycházejí nové architektury CNN.



Obrázek 14 – Schéma jednoduché konvuluční sítě (KARN, 2016)

Na obrázku 14 je zobrazen základní koncept LeNet sítě, která klasifikuje vstupní obrázky do čtyř skupin s určitou pravděpodobností. Součet pravděpodobností musí být 1. Na obrázku 12 je vidět, že síť určila s velkou pravděpodobností, že na obrázku je loď. V CNN jsou čtyři hlavní operace: konvoluce, relu, sdružování (pooling), klasifikace. Tyto operace jsou základem každé architektury CNN. Jako vstup je dán obrázek, který si lze představit jako matici pixelů. Výstupy z každé vrstvy se dělí na kanály a podle filtrů (jader) obrázků.

Konvoluce

Konvoluce se snaží získávat informace z obrázků pomocí malého čtverce zvaného jádro nebo filtr, který se posouvá po pixelové mapě obrázku a snaží se ho převádět postupně na menší pixelové mapy. Jádro má v sobě vyplněné hodnoty, které definují jeho vlastnosti. Konvoluce zachovává vztah mezi pixely. Výpočet probíhá násobením maticí jádra a maticí z pixelové mapy obrázku, které jádro jakoby zakrývá. A následně vynásobením multiplikačními vektory, abychom dostali celé číslo, které se zobrazí do nové matice. Nová matice se nazývá Aktivační mapa nebo Budoucí mapa. Konvoluce má následující parametry, které určují její výsledek:

- Hloubka – Je označení pro počet filtrů použitých pro konvoluci. Pro každý filtr je vytvořena nová Aktivační mapa.
- Posun – Je označení o kolik se posouvá filtr na vstupní pixelové mapě. Když je posun 1, posouvá se o 1 pixel a atd.
- Zero-padding – Určuje, jestli je vstupní pixelová mapa doplněna po okraji nulami pro použití filtrů na hraniční oblasti vstupních dat. Přidáním Zero-

padding, potom říkáme, že se jedná o širokou konvoluci. Když nepoužíváme Zero-padding potom mluvíme o úzké konvoluci.

Relu

Relu operace vkládá do modelu nelinearitu, operace se provádí po operaci konvoluce pro každý pixel. Ten, který má zápornou hodnotu podle dané funkce, je nastaven na nulu, ostatní hodnoty jsou zachovány, tak jak přišly ze vstupu. Místo Relu operace se občas používají funkce tanh nebo sigmoid, ale většinou má Relu lepší výsledky.

Sdružování

Sdružování pomáhá zmenšovat velikost map, ale snaží se zachovávat nejdůležitější vlastnosti obrázků. Je mnoho způsobů, jak může sdružování probíhat. Nejpoužívanější jsou Maximální sdružování, kde se vybírá maximální hodnota z daného výběru, a Průměrné sdružování, které vybírá průměrnou hodnotu.

Klasifikace

Poslední vrstvy jsou propojeny tak, že každý uzel předcházející vrstvy je spojen s každým uzlem finální vrstvy. Jako klasifikační funkce se používá funkce SoftMax, která vezme libovolný vektor hodnot a rozdělí ho na vektor hodnot mezi nula a jedna. Kde je největší hodnota, tam je přiřazen obrázek.

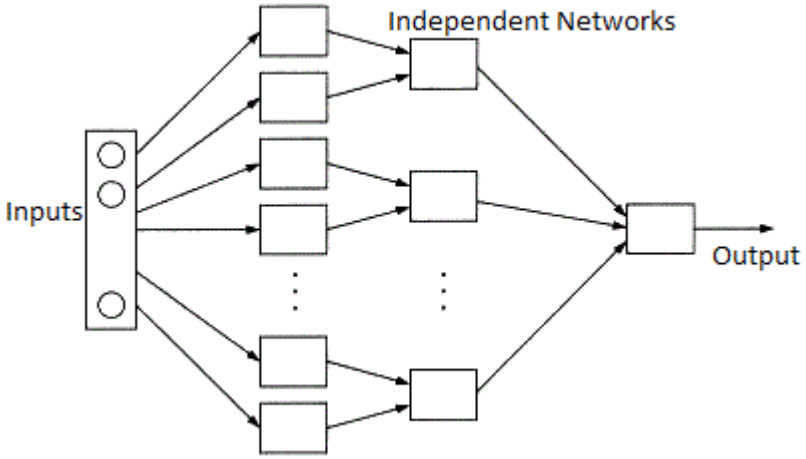
Učení CNN probíhá následujícím algoritmem:

1. Inicializuje filtry a nastaví náhodně parametry a váhy sítě
2. Vezme první obrázek a provede ho sítí.
3. Spočítá se celková chyba
4. Provede se zpětná propagace a aktualizují se váhy a hodnoty filtrů sítě
5. Opakujeme kroky 2-4 se všemi obrázky

Během učení se nemění architektura sítě jako počet a velikost filtrů a počet vrstev. Mění se pouze hodnoty filtrů a váhy mezi uzly (KARN, 2016).

3.2.6 Modulární neuronová síť

Modulární neuronové sítě se skládají z dílčích sítí různých typů, které řeší své úkoly a navzájem se informují o průběhu svého výpočtu. Výsledky výpočtů si posílají mezi sebou. Výhodou modulárních sítí je, že dokážou výpočetně složitý proces rozložit a zjednodušit ho. Na obrázku 16 je vidět, jak může vypadat struktura sítě. Modulární sítě se v poslední době rozšiřují, zejména v oblasti biologie, psychologie a informatiky (MALADKAR, KISHAN, 2018).



Obrázek 15 – Schéma modulární sítě (MALADKAR, 2018)

3.3 Bayesovské sítě

Bayesovské sítě jsou tvořeny na základě znalostí Bayesovy věty o podmíněných pravděpodobnostech. Jsou to metody sloužící k výpočtům pravděpodobnosti jevů, i přesto jsou používány v souvislosti se strojovým učením a dobýváním znalostí.

Bayesovský klasifikátor

Bayesův vzorec podmíněné pravděpodobnosti říká, platí-li hypotéza H za předpokladu, že je splněn jev E vzorec $P(H|E) = \frac{P(E|H)*P(H)}{P(E)}$

Předpokládaná pravděpodobnost *P(H)* říká, jaké zastoupení má hypotéza v daných třídách bez znalostí dalších informací. Podmíněná pravděpodobnost *P(H|E)*, říká jaká

je pravděpodobnost hypotézy H, když víme, že nastal jev E. *P(E)* představuje pravděpodobnost pozorování jevu E, jinak označováno jako evidence jevu E.

Hypotéz, které jsou na výběr, je většinou více. Pro nás je zajímavá ta, která pro danou evidenci bude nejpravděpodobnější. Pro hypotézy H_i i= 1..n, spočítáme $P(H_i \mid E)$ a z nich vybíráme hypotézu, která má největší pravděpodobnost evidence. $H_{map} = H_j \, právě \, když \; P(H_j|E) = \; max_i \frac{P(E|H_i * P(H_i)}{P(E)}$

Nás zajímá pouze to, pro která H_i je hodnota evidence pravděpodobnosti největší, na rozdíl od konkrétní hodnoty, dá se vzorec upravit na $H_{map} = H_j \, právě \, když \; P(E|H_j) * P(H_j) = \; max_i P(E|H_i) * P(H_i)$

Dále se dá předpokládat, že všechny hypotézy H_i jsou stejně pravděpodobné, najdeme tak hypotézu, která má největší věrohodnost $H_{ml} = H_j \, právě \, když \; P(E|H_j) = \; max_i P(E|H_i)$ (HASSANKASHI, 2018).

Bayesovské sítě

Bayesovské sítě reprezentují znalosti o částečně nezávislých evidencích a využít je při usuzování. Vycházejí ze znalostí Bayesovské klasifikace. Bayesovská síť je acyklický orientovaný graf, kde uzly představují náhodné veličiny a hrany mezi nimi představují pravděpodobnosti na základě distribuční funkce P(u| Předci(u)) , kde Předci(u) reprezentují všechny uzly, z kterých vycházejí hrany do uzlu u (BERKA, 2005b).

V těchto sítích se objevují dva druhy znalostí. Znalosti o pravděpodobnostech ohodnocení atributů (určení vah hran mezi uzly) a znalost o struktuře grafu (určit jak budou spojené uzly).

Jsou dva způsoby jak získávat znalosti z dat.

1. Vzít ověřenou strukturu a pomocí dat určovat pouze podmíněné pravděpodobnosti.
2. Z dat vytvořit strukturu sítě a určit i podmíněné pravděpodobnosti.

3.4 Diskriminační analýza

Strojové učení pomocí Diskriminační analýzy vychází ze statistické metody Diskriminační analýzy. Řeší úlohy, kdy klasifikuje jevy do předem zadaných tříd.

Při Diskriminační analýze, je předpoklad že ke každé třídě $c_i, i = 1, .., N$ existuje diskriminační funkce f_i, že platí $f_i(x) = \max_j f_j(x),\ j = 1, .., N$ právě když příklad $x[x_1, .., x_m]$ patří do třídy c_i (BERKA, 2005c).

3.4.1 Lineární diskriminační analýza

Lineární diskriminační analýza je klasifikační metoda, která produkuje jednoduché matematické modely, které jsou stejně dobré jako složitější metody.

V případě lineární diskriminační analýzy má diskriminační funkce lineární tvar $f_i = q_{0l} + q_{1l}x_1 + .. + q_{mj}x_m$ (SAYAD, 2018).

3.4.2 Kvadratická diskriminační analýza

Kvadratická diskriminační analýza má diskriminační funkci v kvadratickém tvaru $f_i = -\frac{1}{2}(x - \mu_i)^T C_i^{-1}(x - \mu_i) - \frac{1}{2}\ln|C_i| + \ln P(C_i)$, kde C_i je kovarianční matice pro třídu i, $|C_i|$ znamená determinant kovarianční matice C_i a $P(C_i)$ je předchozí pravděpodobnost třídy i (SAYAD, 2018).

3.5 Množina rozhodovacích pravidel

Rozhodovací pravidla se používají stejně jako rozhodovací stromy pro klasifikaci vstupních dat do správných tříd. Pravidla mají konstrukci: IF podmínka THEN třída, kde podmínka je tvořena ze vstupních dat. Když je podmínka splněna, je vstupním datům přiřazena třída.

Jedná se o algoritmus typu učení s učitelem, kde je snaha naučit se, přiřazovat vstupním datům správné třídy. V kapitole 3.1 bylo řečeno, že je možné vytvořit rozhodovací pravidla z rozhodovacích stromů. V této kapitole budou ukázány jiné metody tvorby rozhodovacích pravidel.

3.5.1 Pokrývání množin

Algoritmus pokrývání množin nazývány jako Odděl a panuj spočívá v principu nalezení pravidel, která pokrývají nějaké příklady hledaného konceptu. Tyto příklady oddělí od jiných příkladů téhož konceptu i od příkladů jiné třídy. Z toho plyne, že v prostoru atributů se postupně vybírají oblasti, které spadají pod jednu třídu. V grafickém znázornění mají nalezené oblasti algoritmem pokrývání množin stejně jako v případě rozhodovacích stromů podobu mnohorozměrných hranolů rovnoběžných s osami. To vypovídá o tom, že mají stejnou vyjadřovací sílu (BERKA, 2005d).

Algoritmus pokrývání množin

1. Najdi pravidlo, které pokrývá nějaké pozitivní příklady a žádné negativní.
2. Odstraň pokryté příklady z tréninkové množiny.
3. Pokud v tréninkové množině zbývají nějaké nepokryté pozitivní příklady, přejdi k bodu 1, jinak skonči.

3.5.2 Rozhodovací seznam

Rozhodovacím seznamem se rozumí uspořádaný soubor pravidel typu IF podmínka THEN třída, ELSE IF negace všech podmínek předcházejících pravidel THEN třída, kde v závěrech THEN se můžou objevovat různé třídy. Uspořádání zde spočívá v tom, že v každé podmínce ELSE IF se implicitně skrývá negace všech předchozích podmínek předešlých pravidel. Pravidla už nejsou navzájem nezávislá (BERKA, 2005d).

3.5.3 Pravděpodobnostní pravidla

Algoritmy v metodách 3.5.1 a 3.5.2 hledaly pravidla tvořená pouze předpokladem a závěrem. Existují ale i algoritmy, které pravidlům přiřazují různou váhu nebo pravděpodobnost. Jedna metoda jak tato pravidla vytvářet je pomocí systému ITRule. Pravidla vytvořená tímto systémem mají tvar IF podmínka THEN Suc(p), kde podmínku tvoří kombinace kategorií, Suc je jedna kategorie a p je podmíněná pravděpodobnost cíle, nastane-li předpoklad. Každé pravidlo se vypočítává na základě vzájemné informace, nazývané jako j-míra a definovanou vzorcem $j = (Suc, Ant) =$

$P(Suc|Ant) * log_2\left(\frac{P(Suc|Ant)}{P(Suc)}\right) + P(\neg Suc|Ant) * log_2\left(\frac{P(\neg Suc|Ant)}{P(\neg Suc)}\right)$ (BERKA, 2005d).

3.6 Genetické algoritmy

Genetické algoritmy patří do oblasti umělé inteligence, která je inspirována biologií. Nejčastěji se používají k řešení optimalizačních úloh. Neuronové sítě se zde využívají v případech, kde je těžké nalézt řešení analyticky. Vlastnosti živých organismů jsou zakódovány v genetické informaci uložené v sadě buněčných struktur, nazývaných chromozomy. Chromozomy obsahují genetickou informaci vyjádřenou v řetězci DNA. Řetězec je složen z jednotlivých úseků, které se nazývají geny. Zjednodušeně se dá říci, že konkrétní podoby genů a kombinace definují, jaké vlastnosti bude mít konkrétní jedinec. Podle Darwinovy evoluční teorie jedinci s nejlepší genetickou výbavou pro dané prostředí přežívají a rozmnožují se. Postupem času jsou vybrány nejúspěšnější populace.

Genetické algoritmy mají tyto základní pojmy: jedinec, populace, generace, účelová funkce a fitness jedince. Jedinec v generických algoritmech je reprezentován jako řetězec binárních hodnot, kde každá hodnota vyjadřuje, jestli má jedinec danou vlastnost (gen) nebo ne. Populace se skládá z množiny jedinců. Generací nazýváme aktuální populaci jedinců. Účelová funkce provádí výpočet z binárních hodnot do reálných čísel. Schopnost jedince definuje hodnota jeho účelové funkce. Platí, že když je jeho hodnota účelové funkce nižší, jedinec se více hodí do řešení daného problému. Fitness jedince určuje potenciál v daném prostředí vůči ostatním jedincům. Princip práce genetických algoritmů je postupná tvorba generací různými metodami (HOLČÍK a KOMEDA, 2015).

3.6.1 Metody tvorby nové generace

Pro vytváření nových generací a hledání, jestli jsou pro daný problém nejlepší, je zapotřebí mít možnost generovat nové jedince a vybírat ty nejlepší pro tvorbu nových generací. K tomu slouží operátory selekce, křížení, mutace.

3.6.1.1 Selekce

Operátor selekce vybírá jedince, kteří budou dále rozmnožováni. K výběru jedinců se používají následující metody.

Ruletová selekce

Z výběru o N jedincích je vybrán i-tý jedinec s pravděpodobností $p(i) = \frac{F_i}{\sum_N F_j}$. Očekávaná hodnota výběru jedince se rovná podílu kvality daného jedince vůči průměru jedinců v celé populaci. $EV(i) = \frac{F_i}{\frac{\sum_N F_j}{N}} = \frac{N*F_i}{\sum_N F_j}$. Očekávaný výsledek výběru jedince je závislý na kvalitě vyjádřeným Fitness funkcí proti zbytku populace. Z této metody vyplývá, že od začátku jsou vybíráni silnější jedinci, z čehož plyne, že se konverguje rychle k řešení. Nevýhoda je, že se dosti omezuje prostor možných řešení. Dané řešení nemusí být nejlepší.

Seřaďovací metoda

V seřaďovací selekci seřadíme jedince v populaci podle hodnot fitness funkcí. Posledního člena populace ohodnotíme hodnotou $H > 0$. $H = EV(N)$ očekávané hodnoty zbytku populace určíme podle rovnice $EV(i) = D + (H - D)\frac{i-1}{N-1}$, kde $D = EV(1)$ prvního jedince v řadě. Vyběrem posledního jedince populace je snaha zamezit předchozímu případu ruletové selekce, kdy jsou vybírání pouze nejlepší jedinci. Nevýhodou je, že ztrácíme informaci o skutečných kvalitách jedinců.

Selekce lineárním a exponenciálním výběrem

Na začátku seřadíme jedince od nejhoršího k nejlepšímu podle hodnoty fitness funkce. Pravděpodobnost výběru i-tého jedince závisí jenom na indexu i v seřazené posloupnosti. Funkce je buď lineární nebo exponenciální. Lineární je dána předpisem

$p(i) = \frac{1}{N}(n^- + (n^+ - n^-))\frac{i-1}{N-1}$, kde poměr $\frac{n^-}{N}$ a $\frac{n^+}{N}$je pravděpodobnost výběru nejhoršího a nejlepšího jedince. Exponenciální předpis je $p(i) = \frac{c^{N-i}}{\sum_N c^{N-j}}$. Základ exponentu c se volí z intervalu (0,1). Hodnotou tohoto parametru lze zvyšovat či snižovat selektivitu. Selekční algoritmus s exponenciálním výběrem paří k hojně používaným.

Boltzmanův výběr

Boltzmanův výběr pracuje s proměnnou T, která se mění v čase průběhu jednotlivých generací. Na začátku je T nastavena vysoká a postupně klesá k nule. To způsobuje, že ke konci algoritmu jsou vybírání pouze silní jedinci, kteří se rozmnožují.

Další typy selekce

V předešlých metodách se předpokládalo, že po vytvoření nové generace předcházející zaniká, ale ne ve všech metodách tomu tak je. Za zmínku stojí metoda turnajové selekce, kdy se utvoří náhodně dvojice a lepší jedinec postupuje do reprodukce. Další metoda je elitismus, kde nejlepší jedinci postupují společně s potomky do další generace. Metoda ořezávání rozděluje populaci na více částí a do reprodukce postoupí pouze jedinci z nejúspěšnější části.

3.6.1.2 Křížení

Operátor selekce vybírá v generaci jedince, kteří jsou určeni k reprodukci a založení nové generace. Operátor křížení z právě vybraných jedinců vytváří nové potomky. Nejčastěji se používá jednobodové křížení, kdy se od určité pozice genu zbývající části řetězce vymění. Existuje spousta modifikací metody křížení. Například můžeme křížit více než dva rodiče, po křížení můžeme rozhodovat, jaký potomek postoupí a atd.

3.6.1.3 Mutace

Operátor mutace přináší do tvoření potomků prvek náhody, a umožňuje uniknout už z neperspektivního vývoje jedinců. Mutace je ve většině případů realizována, jako jednobodová změna, kdy se náhodně změní jeden gen v řetězci (HOLČÍK a KOMEDA, 2015).

3.6.2 Podobnost generických algoritmů s neuronovými sítěmi

Generické algoritmy se používají i pro optimalizaci neuronových sítí. Nejčastěji se používají pro úpravu topologie sítě a nastavení jejich vah. V topologii neuronových sítí ovlivňují počet vrstev, neuronů a spojů. Jako vstup jim je dána buď minimální struktura sítě a generické algoritmy přidávají a zlepšují síť. Nebo jim je dána komplexní síť a ony ji zjednodušují při zachování stejné výkonnosti sítě. Pro optimalizaci vah neuronové sítě se reálné hodnoty převádějí na binární čísla pro snazší počítání generického algoritmu. Nevýhodou je, že jsme schopni zachytit jenom omezené hodnoty vah. Při prodlužování řetězce se výpočet zpřesňuje, ale taky zpomaluje.

4 Strojové učení pro tagování hudebních skladeb

Zvuk můžeme reprezentovat v základní podobě mnoha způsoby. Zde jsou zmíněny dva hlavní způsoby, kdy se převádí zvukové vlny na spojitý signál, z kterého se dále dá převádět na data, aby jim rozuměl počítač. K tomuto účelu se využívá časové a frekvenční zobrazení. Časové zobrazení udává velikost amplitudy v čase. Používá se k němu amplitudová modulace, která v čase vytváří křivku vstupního zvuku. Frekvenční zobrazení udává velikost každé frekvence zvuku v daném časovém úseku.

4.1 Hudební data

Pro porovnání vědeckých studií většina vědců používá datovou množinu z GTZAN Genre Collection, která obsahuje 1000 skladeb s žánry Blues, Classical, Country, Disco, Hip-Hop, Jazz, Metal, Pop, Reggae, a Rock. Skladby jsou předzpracovány a mají délku 30 vteřin (LEBEN, 2015).

Pro reálné využití modelů je nutné reálné písničky třeba předzpracovat, aby se v modelech porovnávaly vypovídající hodnoty. Není dobré porovnávat začátky skladeb, které jsou zkreslené vůči zbytku písničky. Čím kvalitnější dokážeme zajistit hudební data, tím je větší šance, že bude dosaženo lepších výsledků.

4.2 Způsoby reprezentace hudebních dat

Způsobů, jak reprezentovat zvuk v digitální podobě pro strojové učení, je mnoho. Lze zmínit metody Chroma nebo Chords. Níže jsou uvedeny tři nejčastější způsoby jak reprezentovat zvuk vhodně pro strojové učení hudebních skladeb.

4.2.1 Rychlá Fourierova transformace (Fast Fourier transform, FFT)

Fourierova transformace nám umožňuje převádět mezi časovým a frekvenčním zobrazením. K tomu se využívá harmonických signálů. K převodu daných bodů mezi časovým a frekvenčním zobrazením na konečné intervaly byla vymyšlena Diskrétní Fourierova transformace, která převádí posloupnosti mezi sebou. Nechť máme

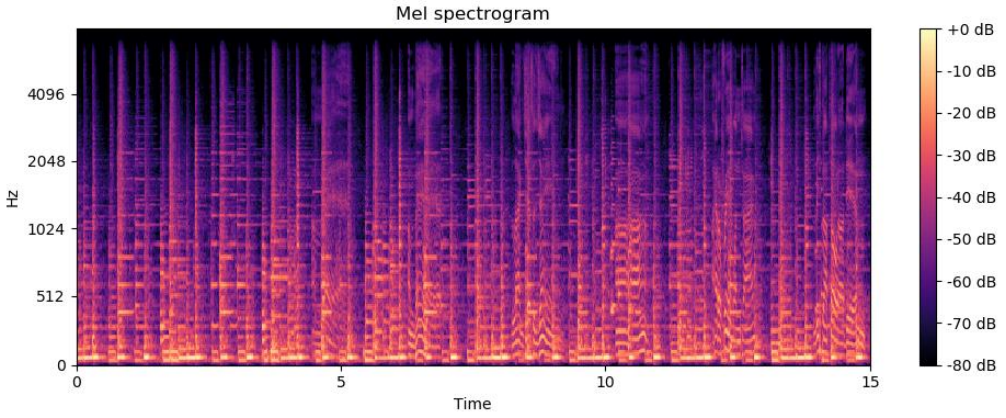
posloupnost komplexních čísel nazývanou X_n. Po provedení Diskrétní Furierovy transformace dostaneme posloupnost Y_n, jejíž body jsou definované $Y_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{2\pi i}{N}kn} = \sum_{n=0}^{N-1} x_n * (\cos\left(\frac{2\pi kn}{N}\right) - i * \sin(\frac{2\pi kn}{N}))$ (MAUTNER, 2014).

Nevýhodou Diskrétní Fourierovy transformace je, že výpočet je poměrně náročný. Proto byla dále vymyšlena metoda Rychlá Fourierova transformace, která pracuje v časové složitosti *O(n log n)*. Je velmi používána napříč obory. V našem případě nám dá graf časové zobrazení, na kterém se dají provádět výpočty ve strojovém učení. Tato metoda je základem v mnoha dalších metodách, které ji využívají k pomocným výpočtům (SMITH, 2003).

4.2.2 Spektrogram

Spektrogram zobrazuje vizuální podobu frekvencí zvuků v čase. Využívá se přitom Short-Time Fourier Transform (STFT), která počítá po krátkých časových úsecích FFT. Z výstupu FFT se provede mapování podle velikostí amplitudy a frekvence na dané barvy. A z těchto všech dílečků je poskládán jeden obrázek, kterému se říká spektrogram (PRAHALLAD, 2003).

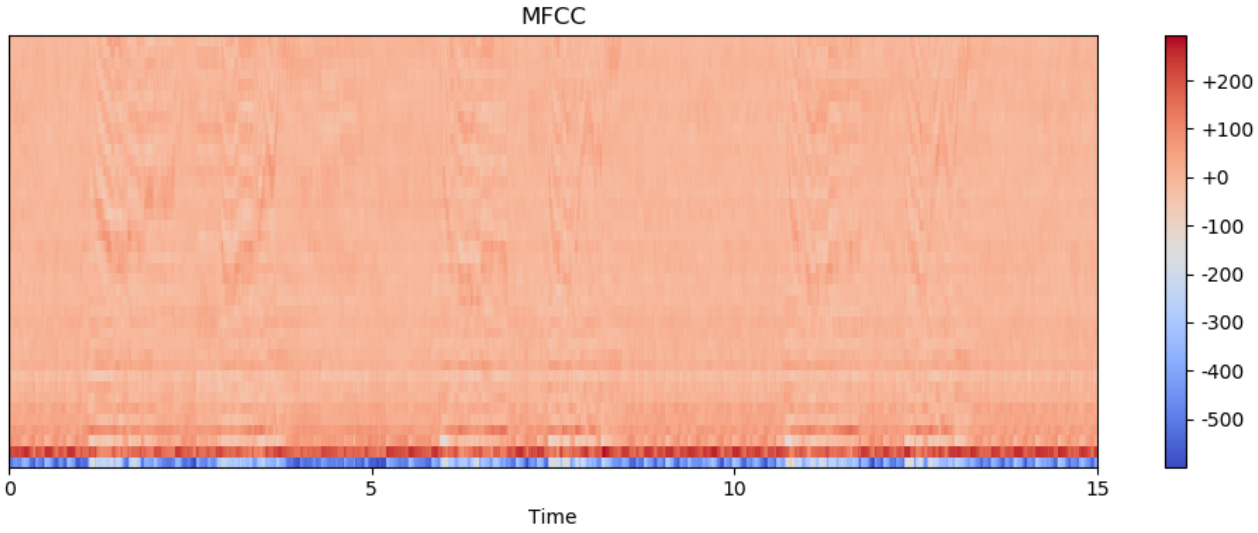
Na následujícím obrázku 16 je uveden příklad spektrogramu jedné skladeb z tréninkové množiny.



Obrázek 16 – Spektrogram (vlastní zpracování)

4.2.3 Mel-frequency Cepstral Coefficient (MFCC)

MFCC se vytváří ze začátku podobně jako spektrogram, ale dále se nad ním vytvářejí další operace. Nad každým dílkem spektrogramu je provedena operace Mel-filtr a přepočet na Mel-frekvenci. Dále je spočítán logaritmus z Mel-frekvence. Nakonec se provádí Diskrétní kosinová transformace (PERQLÄ, 2015). Existuje několik typů diskrétní kosinovy transformace. Sloučením dílků dostaneme výsledný obraz znázorňující MFCC pro danou skladbu (LYONS, 2012). Na obrázku 17 je uveden MFCC diagram pro jednu písničku z množiny GTZAN.



Obrázek 17 – MFCC (vlastní zpracování)

4.3 Metody strojového učení pro tagování hudebních dat

V kapitole 3 je zmíněno, že pro tagování hudebních skladeb se používají hlavně následující metody: K-nejbližších sousedů, Náhodný les, Naive Bayes, Rozhodovací pravidla s užitím Support vector machines, Neuronové sítě. V následujících podkapitolách bude ukázáno, jak se dané technologie přizpůsobují k řešení problému tagování hudebních skladeb.

4.3.1 K-nejbližších sousedů

Metoda K-nejbližších sousedů pro tagování hudby funguje tak, že vstupní data (písničky) jsou pomocí metody FFT převedena na data, která můžeme v jednotlivých bodech porovnávat. K porovnání dvou skladeb se používají metody Kullback-Leiblerova divergence, Minkowskiho vzdálenost nebo Manhattanská vzdálenost. Z K-nejbližších skladeb, které jsou nejčastěji podobné dané skladbě, se odvodí výsledný žánr písně. Musí být předem učena data, se kterými se poté dané skladby porovnávají.

4.3.2 Náhodný les (modifikace rozhodovacích stromů)

Metoda Náhodný les vychází z metody Rozhodovacích stromů. Je to soubor klasifikačních stromů postavený tak, aby se minimalizovalo zkreslení a korelace mezi jednotlivými stromy. Každý strom je tvořen z jiné množiny tréninkových dat. Pro postavení stromu z tréninkových dat je zhruba 1/3 dat nahrazena a náhodně odstraněna. Tato odstraněná data se nazývají OOB (out of bag). Každý strom je stavěn v největší možné míře. Opakováním tohoto postupu dostaneme N-stromů, takzvaný les. Počet opakování vytváření se určuje, dokud přidáváním stromů nevzniká větší chybovost. Chyba klasifikace na tréninkových datech se počítá tak, že se vezme počet správných rozhodnutí z dat OOB (KURSA, RUDNICKI,WIECZORKOWSKA, KUBERA a KUBIK-KOMAR, 2009).

4.3.3 Naive Bayes (modifikace Baysovské metody)

Problém hledání správných žánrů si můžeme matematicky představit jako $\hat{g} = \arg\max_{g \in G} P(g|\bar{X})$, kde $\bar{X}$ je vstupní proměnná pro hudební data, G je množina žánrů. Snaha je s největší pravděpodobností správné žánry přiřadit daným skladbám. To se dá pomocí Bayesových pravidel interpretovat jako $\hat{g} = \arg\max_{g \in G} \frac{P(\bar{X}|g)*P(g)}{P(\bar{X})}$

Pravděpodobnost $P(g|\bar{X})$ můžeme vyjádřit jako $P(g|\bar{X}) = \frac{P(\bar{X}|g)*P(g)}{\sum_{g \in G} P(\bar{X}|g)*P(g)}$. Tím dostaneme celý vzorec pro výpočet pravděpodobnosti. Nevýhodou Naivní Bayseovy

metody je, že předpokládá nezávislost vstupních atributů, což se ve skutečnosti tak často neděje a tím vzniká větší chybovost (SILLA JR, KOERICH a KAESTNER, 2018).

4.3.4 Rozhodovací pravidla s užitím Support vector machines

Algoritmus rozhodovacích pravidel se dá využít pro klasifikaci hudebních žánrů za podpory statistického algoritmu Support vector machines (SVM) následujícím způsobem. Základní myšlenka tohoto algoritmu pro užití je, že využívá různé klasifikační funkce na dělení hudebních dat. Lze zmínit např. Beat Spectrum, která určuje tempo písní, LPC-Derived Cepstrum, která se snaží najít opakování v písních. Zero Crossing Rate měří rychlost, se kterou dochází k nulovému přechodu. Dále to mohou být metody Spectrum power a MFFC. Dále probíhá strojové učení algoritmem Support vector machines, který se učí najít nejlepší klasifikační funkci pro dvě skupiny žánrových dat. Jak Support vector machines pracuje, je popsáno v dalším odstavci. Jako výstup dostaneme rozhodovací strom, podle kterého dělíme hudební data do příslušných žánrů. Algoritmus pracuje s chybovostí kolem 7%.

Algoritmus Support vector machines jako vstup dostává trénovací data $X_1, .., X_n$ a jejich žánry $Y_1, .., Y_n$, kde X_i patří R^n, Y_i patří {-1,+1} a snaží se rozdělit data do dvou tříd. Základní myšlenka je transformovat vstupní vektory do prostor, kde je vidět přesněji, do jaké skupiny data patří, čili do prostor vyšší dimenze. Pro konstrukci nelineárního SVM klasifikátoru je potřeba funkce *K(x,y)* nazývaná jako jádro funkce SVM klasifikátoru. SVM klasifikátor je $f(x) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b)$. SV algoritmus může konstruovat různé kernelové funkce (XU, MADDAGE, SHAO, CAO a TIAN, 2003):

1) Funkce s polynomiálním jádrem stupně d

   $K(x, y) = (< x, y > +1)^d$

2) Funkce s radiálním základem a Gaussovým jádrem šířky c > 0

   $K(x, y) = \exp(\frac{-||x-y||^2}{c})$

3) Neuronové sítě s aktivační funkcí tanh

$K(x, y) = \tanh(k < x, y > +\mu)$, kde k a μ značí zisk a posun

4.3.5 Neuronové sítě

Neuronové sítě se v poslední době stávají používané ke klasifikaci nebo predikci zvukových dat a v praxi nahrazují předešlé metody. To zejména kvůli svým lepším výsledkům. Níže jsou ukázány dva typy neuronových sítí při použití na reálných modelech pro klasifikaci hudebních dat.

4.3.5.1 Konvoluční neuronové sítě

Tagování hudebních skladeb s pomocí konvolučních neuronových sítí probíhá tak, že se snažíme postavit model konvoluční neuronové sítě a na něm trénujeme skladby, které jsou předzpracovány a převedeny na jeden z možných typů reprezentace pro počítačové zpracování, jak bylo zmíněno v kapitole 4.1 Způsoby jak reprezentovat hudební data.

Na následujícím příkladu ukážeme, jak může taková konvoluční neuronová síť pro tagování hudebních skladeb vypadat.

Následující příklad používá pro reprezentaci skladeb MFFC, kde jedno časové okno má 23 ms s překrýváním 50%. Struktura neuronové sítě se skládá z následujících vrstev.

Struktura sítě

1. Vstupní vrstva: 64 * 256 neuronů, kde 64 znázorňuje rozsah hodnot MFFC a 256 počet časových oken MFFC
2. Konvoluční vrstva: 64 hloubka, filtry o rozměrech 3 * 3 s posunem 1
3. Sdružovací vrstva: 2 * 4
4. Konvoluční vrstva: 64 hloubka, filtry o rozměrech 3 * 5 s posunem 1
5. Sdružovací vrstva: 2 * 4
6. Klasifikační vrstva: 32 neuronů propojených s každým neuronem z předchozí vrstvy

7. Výstupní vrstva: 10 neuronů propojených s každým neuronem z předchozí vrstvy

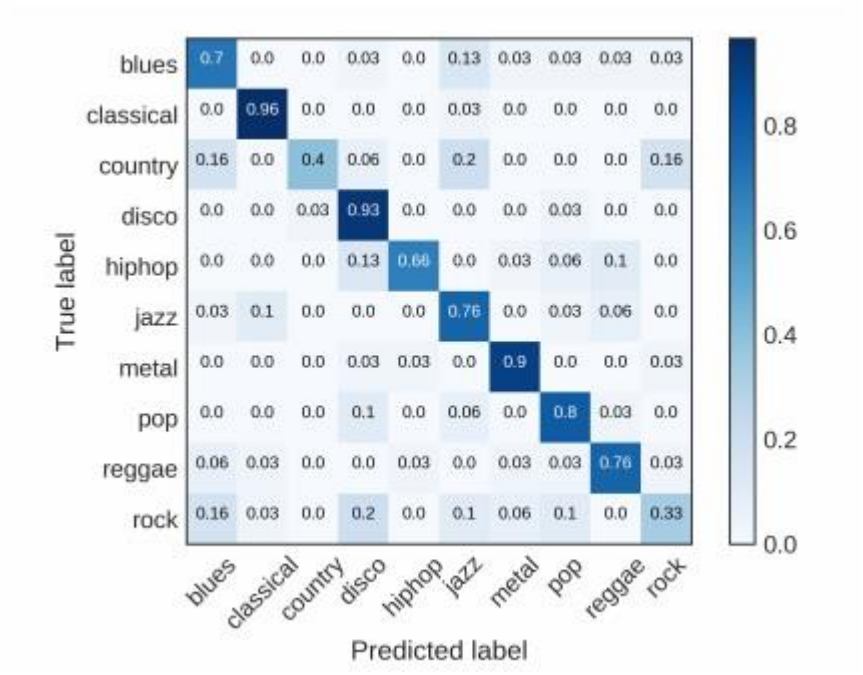
   Ve skrytých vrstvách se používá RELU aktivační funkce, výstupní vrstva používá Softmax funkci.

Algoritmus trénování modelu se trochu liší od obecného algoritmu popsaného v kapitole 3.2.5 Konvoluční neuronové sítě. Algoritmus vybírá náhodně skladby z trénovací množiny a ukončení trénování je provedeno až ve chvíli, kdy nedochází ke zlepšování modelu.

Po ukončení trénování modelu se ještě následujícím algoritmem upravují filtry modelu. Všechny tréninkové skladby jsou rozděleny do tří sekundových segmentů s 10 % překrytím. Všechny části skladeb se spouštějí na vytrénovaném modelu. Jejich průběžné výsledky ukládáme mezi vrstvami. V algoritmu se používá Lasso regrese. Je typem lineární regrese, která využívá smršťování k centrálnímu bodu. Algoritmus probíhá v následujících krocích:

1. Určit úsek vstupních neuronů, který by mohl aktivovat cílový neuron ve specifické vrstvě. $c_{i,j}^{l}$ označuje neuron v rozsahu *i* až *j* neuronů v *l*-té vrstvě.
2. Provést Lasso regresi s vybranou sekcí z MFFC, která bude reprezentována jako vektor a odpovídající aktivaci neuronu $c_{i,j}^{l}$
3. Získané hodnoty z Lasso regrese jsou použity ke změně filtrů.

Takto sestavená síť dosáhla na tréninkově množině 1000 skladeb, které byly v poměru 5:2:3 rozloženy pro trénování, validaci (přepočítání filtrů) a testování výsledků pro deset různých žánrů s následujícími úspěšnosti zobrazené na obrázku 18 (DONG, 2018).



Obrázek 18 – Přehled výsledku z testování konvoluční sítě (DONG, 2018)

Konvoluční neuronové sítě mohou mít různé struktury a vylepšení obecného konceptu pro zlepšení tagování skladeb.

4.3.5.2 Rekurentní neuronové sítě

Využití rekurentních neuronových sítí při klasifikaci hudebních skladeb je velice časté a ve studiích je dosahováno solidních výsledků. Většinou se ale jedná o modely, které nejsou čistě rekurentní, ale kombinovány s konvolučními sítěmi a dalšími upravenými rekurentními sítěmi.

Z článku Recurrent Neural Networks with Attention for Genre Classification (IRVIN, CHARTOCK a HOLLANDER, 2016), který se zabýval čistě rekurentními sítěmi pro klasifikaci skladeb, byly popsány tři hlavní modely, které byly tvořeny z buněk RNN, LSTM a LSTM s vrstvou Soft attention. Používali hudební skladby z GTZAN. Hudební data převáděli na spektrogram tak, že pro každý pětisekundový úsek měli rozsah frekvencí do 4000Hz, který rozdělili na 513 dílku s 22 časovými kroky. Tím dostali 6000 pěti sekundových klipů. Úspěšnost těchto modelů je zobrazena na následujícím obrázku 19. Z obrázku 19 je vidět, že nejlepších výsledků dosáhl model LSTM s dvěma

vrstvami. Výsledná úspěšnost není zas tak velká. Domnívám se, že by výrazně pomohlo, kdyby se tvořily spektrogramy s překrytím.

Model	Test Accuracy
Vanilla RNN: 1 Layer, 125 Dimensional Cell	0.54
Vanilla RNN: 2 Layer, 125 Dimensional Cell	**0.655**
Vanilla LSTM: 1 Layer, 250 Dimensional Cell	0.745
Vanilla LSTM: 2 Layer, 250 Dimensional Cell	**0.79**
LSTM with Attention: 1 Layer, 500 Dimensional Cell	0.7316
LSTM with Attention: 2 Layer, 60 Dimensional Cell	**0.758**

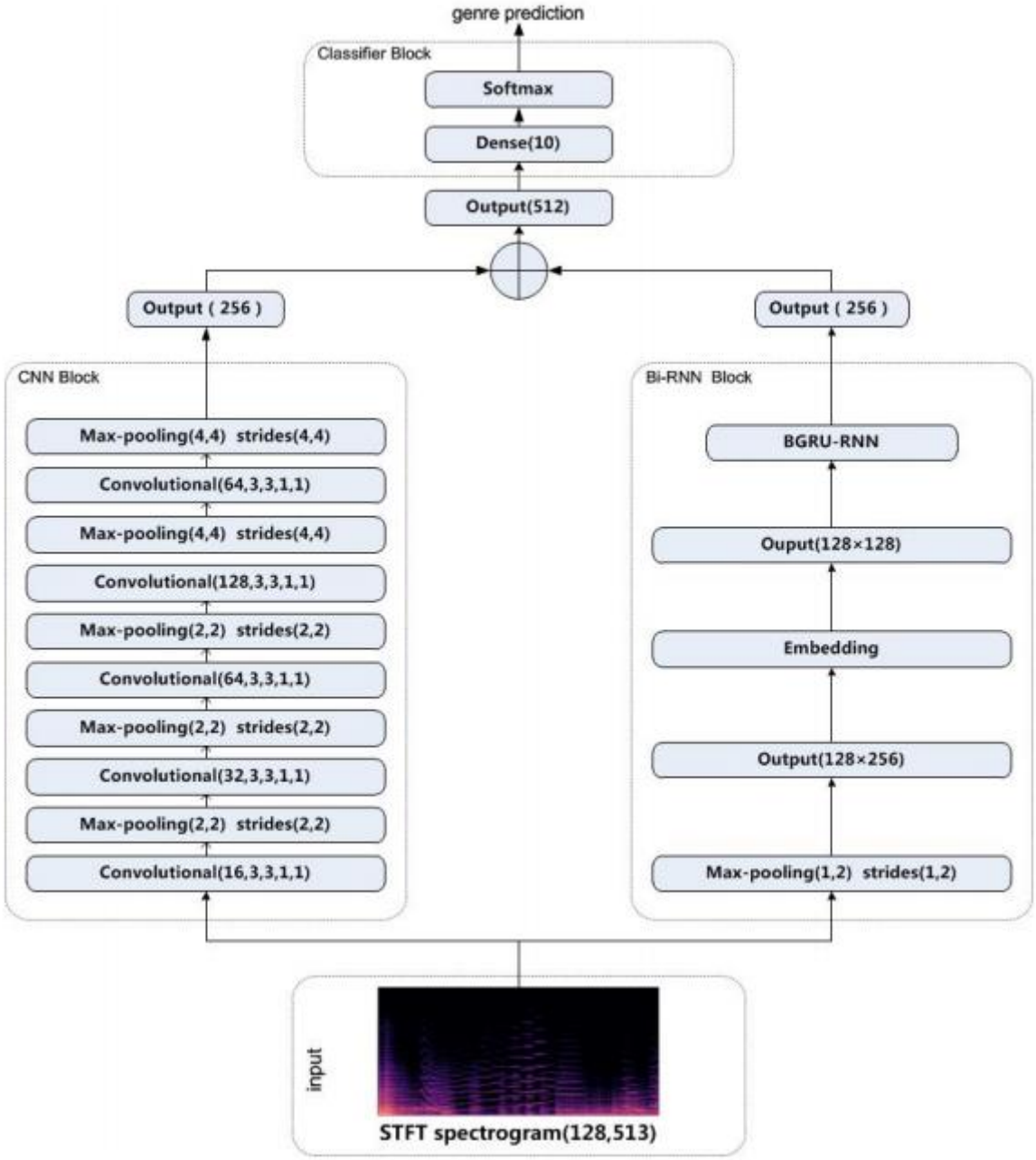
Obrázek 19 – Přehled výsledků z testování rekurentních sítí (IRVIN, CHARTOCK, HOLLANDER, 2016)

4.3.5.3 Hybridní neuronové sítě

Hybridní neuronové sítě bývají tvořeny kombinací několika typů sítí pro zachycení různých vlastností hudebních dat.

Následující model, který je podrobně popsán v článku Music Genre Classification with Paralleling Recurrent Convolutional Neural Network (FENG, LIU a YAO, 2017), je ukázáno, jak se takový model staví.

Model je postaven kombinací konvoluční a rekurentní síti. Konvoluční část deset vrstev, z toho pět je konvolučních a pět sdružovacích, ve skrytých vrstvách je použita operace Relu. Jako výstup dostaneme jednodimenzionální vektor o velikosti 256. Rekurentní část sítě se skládá z tří vrstev, první vrstva sdružování zmenšuje vstup na rozměr 128 * 256, druhá vrstva je před-trénovaná a zmenšuje vstup na 128 * 128. Třetí vrstva využívá jednotky BiGRU. Jako výstup dostaneme také jednodimenzionální vektor o velikosti 256. Výstupy obou částí modelu jsou spojeny do jednoho vektoru. Pomocí funkce Softmax je určen výsledný žánr.

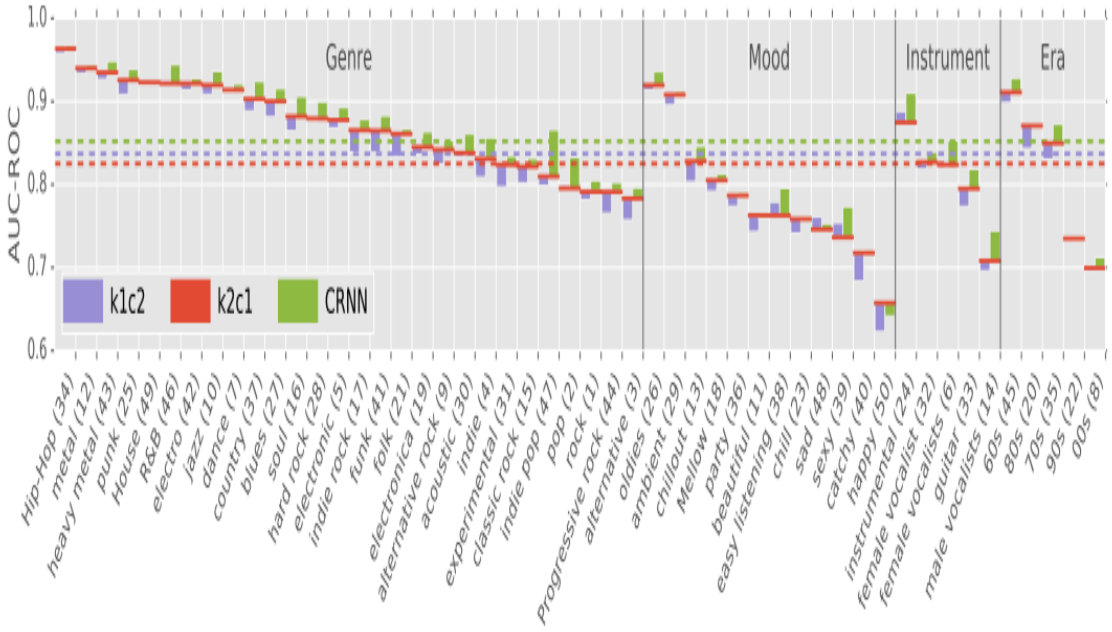


Obrázek 20 – Schéma hybridní neuronové sítě (FENG, LIU, YAO, 2017)

Konvoluční část se stará o získávání prostorových informací ze spektrogramu. Rekurentní část se snaží zajistit souvislosti mezi časovými rámci. K tomu se využívájí buňky BiRNN, které dokáží pracovat jak s časovou informací minulou, tak i budoucí. Následné spojení výstupů obou částí do jednoho vektoru dává kvalitnější představu o dané skladbě. Model byl testován také na hudebních skladbách z GTZAN. Model dosáhl

90.2 % úspěšnosti s jednou vrstvou BiRNN, s dvěma vrstvami BiRNN dosáhl menší úspěšnosti 88 %. (FENG, LIU a YAO, 2017)

Další model, který kombinuje CNN s rekurentními sítěmi (CRNN) je popsán v článku CONVOLUTIONAL RECURRENT NEURAL NETWORKS FOR MUSIC CLASSIFICATION (CHOI, FAZEKAS, SANDLER, CHO, 2016). Srovnává modely konvolučních sítí s různými filtry. CRNN síť je postavena tak, že má čtyři vrstvy CNN sítí a kde konvoluční vrstva má rozměr 3 * 3 a sdružovací vrstvy mají postupně rozměry (2×2)-(3×3)-(4×4)-(4×4). Výstup z poslední CNN vrstvy je přiveden na vstup rekurentní sítě s dvěma vrstvami GRU neuronů. Jako reprezentaci hudebních skladeb používaly modely metodu MFCC. Modely byly trénovány na datové množině obsahující písně z rádia Last.fm o velkosti zhruba 214284 písní s padesáti druhy tagů, kde tagy nebyly jenom žánry skladeb, ale i nálady písní, druhy nástrojů a éra hudební skladby. Na Obrázek 21 19 můžeme vidět výsledky testování CNN modelů a CRNN modelu. Ve většině žánrových případů, dává CRNN model lepší výsledky. Toto testování se liší od ostatních tím, že probíhalo na velkém množství dat, ale klasifikovalo do velkého množství tříd, proto nejde úplně srovnávat výsledky tohoto modelu s předchozími (CHOI, FAZEKAS, SANDLER, CHO, 2016).



Obrázek 21 - Přehled výsledků z testování CRNN síťí - (CHOI, FAZEKAS, SANDLER, CHO, 2016)

4.3.5.4 IdRnn neuronové sítě

IdRnn neuronové sítě jsou jedny z nemladších neuronových sítí. Využívají nezávislosti svých neuronů, které nejsou ovlivňovány dalšími neurony ze stejné nebo vyšší vrstvy. To má pozitivní vliv na klasifikaci písní, kde kratší úseky písniček odpovídající třeba trochu jinému žánru, neovlivňují neurony ze správně klasifikovaných úseků. Tím se chyby nešíří do celé šíře neuronů v jedné vrstvě. V práci Music Genre Classification Using Independent Recurrent Neural Network (WU, SONG, WANG a HAN, 2018) je testován, který byl utvořen z 5 vrstev IdRnn neuronů. Učení modelu probíhalo na datové množiny GTZAN. Na devět-seti písních, které byly rozděleny do devíti adresářů, probíhalo trénování, a sto bylo použito pro testování během trénování. Pro finální testování bylo použito 10 písní z každého adresáře a bylo dosaženo úspěšnosti 96 procent.

5 Charakterizace vybrané metody strojového učení pro tagování skladby

Metodu, která byla vybrána pro stavbu modelu, se podobá metodě z článku Music Genre Classification with Paralleling Recurrent Convolutional Neural Network (FENG, LIU a YAO, 2017), kterou se budeme snažit vylepšit a dosáhnout lepších výsledků. Z předchozích modelů v minulé kapitole dosahoval nejlepších výsledků model postavený z neuronů IdRnn z kapitoly IdRnn neuronové sítě, který měl úspěšnost 96%. Jelikož se jedná o zcela nové poznatky, nejsou ještě podporovány IdRnn neurony v frameworkcích na stavbu modelu. Proto bylo rozhodnuto použít již existujících neuronů, které podporují frameworky.

Hudební data pro trénování modelu byla použita z množiny GTZAN.

V rámci pokusů byly implementovány dvě verze modelů, které se liší akorát jiným převodem z písničky na hudební data. První model převádí každý třiceti sekundový úsek metodou MFCC na příslušný graf a druhý používá spektrogram. Graf bude vstup do trénování modelu. Graf je matematicky zobrazen jako dvourozměrné pole s hodnotami spektra. Parametry MFCC metody jsou následující: časové okno je 2048(velikost okna při FFT), posun časové okna je 1024, počet MFCC je 40, jako Diskrétní cosinova transformace byl zvolen typ dvě. Spektrogram má časové okno taky 2048 a posun 1024

Samotné učení probíhá pomocí metody Hlubokého učení (Deep learning). Hluboké učení je podobor strojového učení. Využívá neuronových sítí s velkým počtem vrstev. Počet vrstev určuje hloubku modelu. Hluboké učení hledá asociace mezi vstupními a výstupními daty. Pro trénování modelu využívá algoritmu zpětné propagace. Čím více máme kvalitnějších dat pro trénování, tím bývá výsledný model lepší. Při učení je možné přes daná data vícekrát projít. (LIANG, 2018) Důležitou součástí hlubokého učení je správná volba chybové funkce (loss function) a zvolené metody optimalizace. Chybových funkcí je několik typů, zde si uveďme nejběžnější (CHANGHAU, 2018):

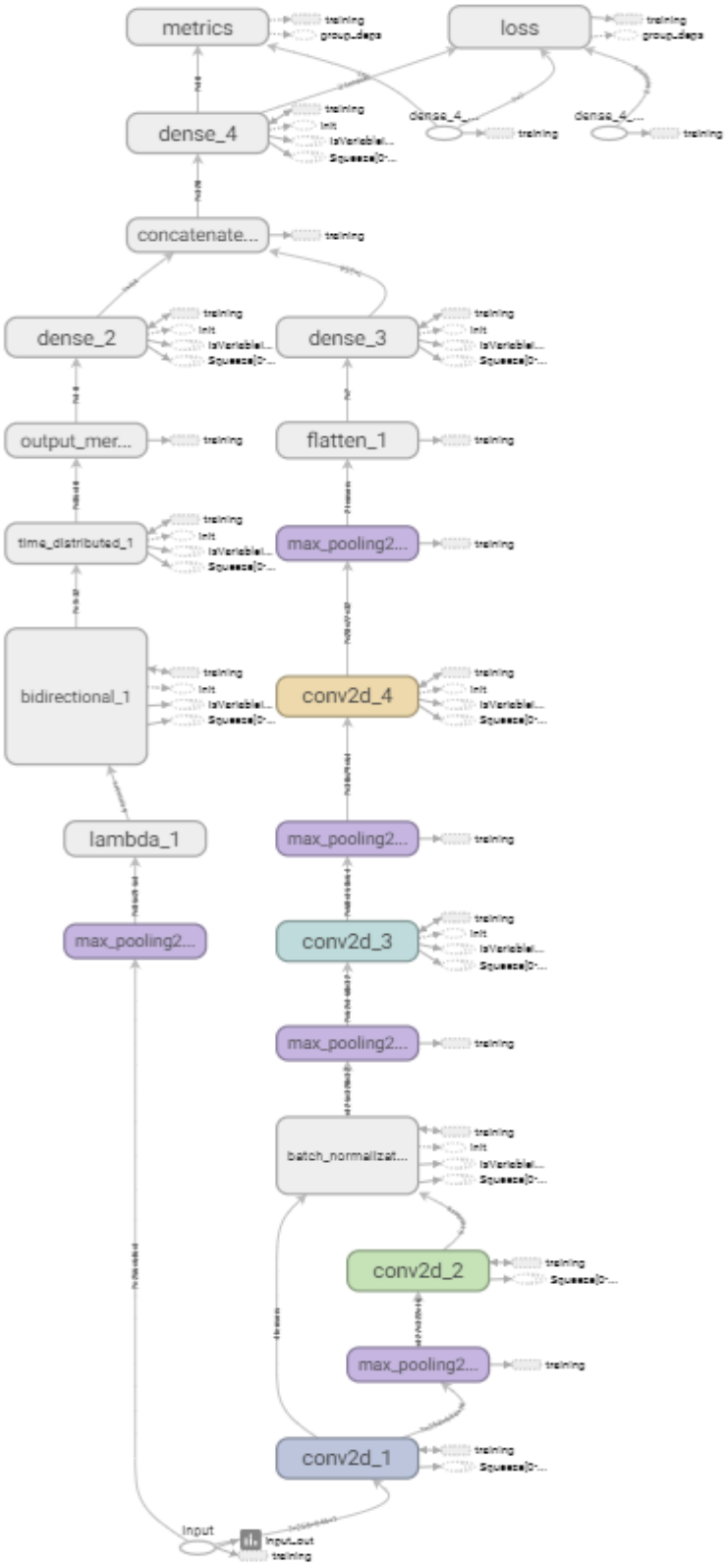
- Průměrná kvadratická chyba (Mean Squared Error, MSE)

- Mean Squared Logarithmic Error (MSLE)
- Mean Absolute Error (MAE)
- L1
- Kullback Leiblerova divergence (KL)
- Categorical cross entropy
- Binary cross entropy
- Poison

Optimalizace je proces hledání parametrů modelu se snahou maximalizovat naše výsledky přesnosti modelu. Nejběžnější metody optimalizace jsou následující (SKALSKI, 2018):

- Stochastic gradient descent(SGD)
- RMSprop
- Adam

Samotný model má následující strukturu zobrazenou na obrázku 22



Obrázek 22 - Graf modelu (vlastní zpracování)

Model se dělí na několik bloků, v pravé části grafu můžeme vidět CNN blok, který tvoří konvoluční neuronová část. Ta se skládá z pěti vrstev sdružování a pěti konvoluce. Konvoluční vrstvy používají aktivační funkci Relu. Na obrázku 23 jsou popsané, jaké mají další parametry vrstvy CNN bloku. V levé části grafu je vidět blok, který je tvořen z Bi-LSTM neuronů, obsahuje taky jednu vrstvu sdružovací. Bi-LSTM vrstva obsahuje 128 neuronů Bi-LSTM buněk. Sdružovací vrstva má parametry 1*1 rozměr okna s posunem 1*1.

	posun	jádro konvulace	rozměr sdružování	počet filtrů
1.Konvoluce	1*1	3*3		16
1.Sdružování	2*2		2*2	
2.Konvoluce	1*1	3*3		32
2.Sdružování	2*2		2*2	
3.Konvoluce	1*1	3*3		64
3.Sdružování	2*2		2*2	
4.Konvoluce	1*1	3*3		32
4.Sdružování	4*4		4*4	

Obrázek 23 – Zobrazení parametrů konvoluční vrstvy verze užití spektrogramu (vlastní zpracování)

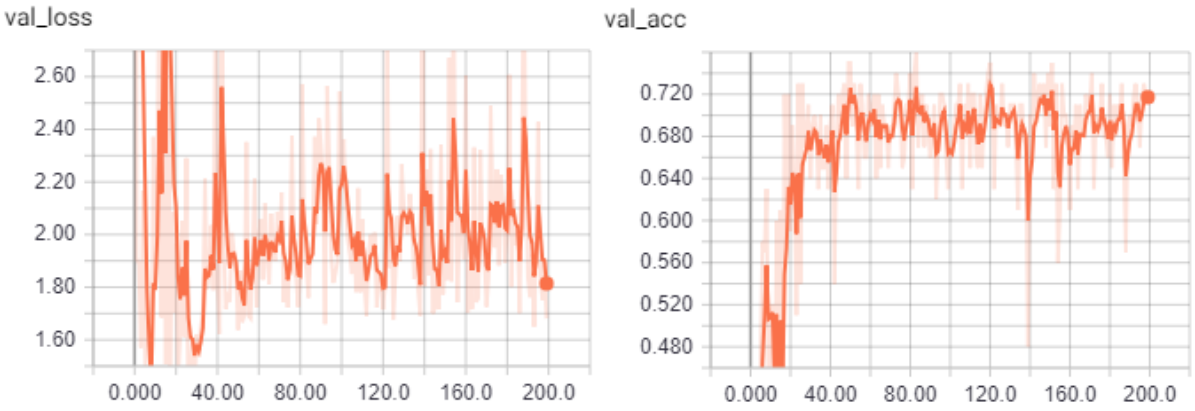
Poslední blok tvoří spojování konvolučního bloku a Bi-LSTM bloku, kde oba bloky jsou převedeny na 256 neuronů a spojeny do jedné vrstvy. Další vrstva má 10 neuronů, která je propojena s každým neuronem z předešlé vrstvy a využívá aktivační funkci Softmax.

Hudební data po provedení MFCC metody na data o dvourozměrném poli, mají rozměr 40 * 646, kde 40 znamená počet MFCC a 646 počet časových dílků. Takové pole je vstupem do modelu. Data pro spektrogram mají rozměr 256 * 646, kde 256 znamená počet mel frekvencí a 646 počet časových dílků.

Nejlepších výsledků modelu bylo dosaženo při použití spektrogramu jako metody na převod dat.

Modely jsou trénovány na množině 900 písní a testovány na 100 písních a opakovaně zlepšovány ve sto cyklech. Model nemá inicializované váhy. Jako optimalizaci využívá metodu RSMSprop. Chybová funkce modelu je Categorical Cross Entropy.

Na obrázku 24 jsou diagramy z průběhu trénování. Grafy vyjadřují příslušné hodnoty po provedení jednoho cyklu trénování na testovacích datech. Pro nejlepší model bylo provedeno trénování ještě v 200 cyklech, ale žádného zlepšení se nedosáhlo. Největší přesnosti na testovacích datech bylo dosaženo po 86 cyklu.



Obrázek 24 - diagramy z trénování dat na testovací množině (vlastní zpracování)

6 Implementace

K implementování modelu neuronových sítí existuje mnoho frameworků. K nejznámějším patří Tensorflow a PyTorch. Další framework je Keras, který má uživatelsky vlídnější api a je nadstavbou nad Tensorflow. Funguje i nad dalšími technologiemi jako třeba Theano. Samotný kód vychází z projektu Music Genre Recognition (KOZAKOWSKI a MICHALAK, 2016), který se zabývá rozpoznáváním hudebních žánrů v aktuálním čase písničky. Základní kostra budování modelu je tedy podobná s tímto projektem. Používání modelu je však jíž odlišné.

6.1 Použité technologie

Tensorflow a Keras

Pro implementování modelu popsaného výše jsem zvolil knihovny Tensorflow s Keras. Tensorflow je celosvětově nejpoužívanější technologií pro vývoj neuronových sítí. Tensorflow vyvíjí společnost Google. Původně byl vyvíjen týmem Google Brain pro vlastní použití Googlu. Následně byl publikován jako opensource. Tensorflow využívá mnoho velkých společností po celém světě. Příkladem je Airbnb, Amd, Uber a spoustu dalších. Keras je uživatelsky přívětivé API nad Tensorflow technologií, ve kterém jde celkem intuitivně vytvářet modely neuronových sítí. V Googlu se rozhodli, že v nové verzi Tensorflow 2 už bude plně implementováno rozhraní Keras. Tensorflow je implementováno ve třech programátorských jazycích Pythonu, C++, CUDA, kde v každém jazyce je implementována jiná část. Tensorflow je spustitelné na mnoho platformách, jako Linux, Windows, Android, macOS a Javascriptu. Často se stává, že spuštění pod OS Win je docela problematické. Toto lze v celku jednoduše vyřešit instalací programu Docker a spouštět Tensorflow v něm. Tensorflow umožňuje k výpočtu používat buď CPU nebo grafické karty, pokud podporují technologii CUDA. Google na svých cloudech používá jednotky TPU(Tensor Processing Units), které jsou přímo vyvinuty pro strojové učení. Použitím TPU jednotek se dosahuje větší rychlosti učení (TENSORFLOW, 2019). Pro průběžné zkoumání výsledků, nám Tensorflow nabízí jejich vlastní program TensorBoard, který je součástí knihovny Tensorflow.

V programu Tensorboard můžeme vidět, jak se náš model učí, jestli se naopak nezhoršuje nebo nestagnuje a není třeba učení ukončit. K tomu využívá nejrůznějších grafů a proměnných. Další užitečná věc co nám TensorBoard nabízí, je grafické zobrazení modelu, kdy vygeneruje celý graf modelu (TENSORBOARD, 2019).

Python

Vlastní zdrojový kód je psán v jazyce Python, konkrétně ve verzi 3.5. Python je moderní programovací jazyk, který je v posledních letech velmi oblíbený. Jeho historie sahá až do roku 1991 a postupně se vyvíjí. Nyní už je vyvinuta verze 3.7.2 a roku 2020 přestane být podporována verze 2.7 (PYTHON SOFTWARE FOUNDATION, 2019a). Pro instalování více projektů s různými závislostmi na knihovnách a verzích Pythonu je použit program Virtualenv (BICKING, 2018). Pro instalování knihoven se používá program Pip (PYTHON SOFTWARE FOUNDATION, 2019b).

Pro verzování kódu a dostupnost z internetu se využívá program Git (CHACON a LONG, 2019) a platforma GitHub (GITHUB COMPANY, 2019).

6.2 Programátorská dokumentace

Program je napsán v jazyce Python pomocí frameworku Tensorflow a Keras. Pro práci s transformací hudebních souborů na datové soubory je použita knihovna Librosa(LIBROSA DEVELOPMENT TEAM, 2019). Program se skládá ze tří částí: příprava dat, stavba a trénování modelu a aplikace modelu.

6.2.1 Příprava dat

O přípravu dat se stará soubor create_data_pickle.py. Skript obsahuje tyto parametry:

- -d, --dataset_path, kde se definuje vstupní hudební adresář.
- -o, --output_pkl_path, kde se definuje výstupní soubor transformovaných dat.

Hlavní metoda souboru je metoda collect_data, která má tři hlavní proměnné x, y, track_paths a pro každou písničku počítá jedinečný index. V poli x má každá písnička svoji datovou reprezentaci po provedení load_track ze souboru common.py. V poli y

má každá písnička uloženo, jakého je žánru. Pole track_paths obsahuje informace o cestách k hudebním souborům daných písniček. Na obrázku 25 je zobrazen zdrojový kód metody collect_data.

```
def collect_data(dataset_path):
    default_shape = get_default_shape(dataset_path)
    x = np.zeros((TRACK_COUNT,) + default_shape, dtype=np.float32)
    y = np.zeros((TRACK_COUNT, len(GENRES)), dtype=np.float32)
    track_paths = {}

    for (genre_index, genre_name) in enumerate(GENRES):
        for i in range(TRACK_COUNT // len(GENRES)):
            file_name = '{}/{}.000{}.au'.format(genre_name,
                    genre_name, str(i).zfill(2))
            print('Processing', file_name)
            path = os.path.join(dataset_path, file_name)
            track_index = genre_index  * (TRACK_COUNT // len(GENRES)) + i
            x[track_index], _ = load_track(path, default_shape)
            y[track_index, genre_index] = 1
            track_paths[track_index] = os.path.abspath(path)

    return (x, y, track_paths)
```

Obrázek 25 – metoda collect_data (vlastní zpracování)

Metoda load_track ze souboru common.py. převádí hudební data na počítačová, aby se s nimi dalo pracovat v neuronové síti. K tomu se používá knihovna Librosa, která obsahuje metodu load. Ta načte data ze souborů a vrací pole časových rámců a rozsah vzorkovacích frekvencí. Pro výpočet spektrogramu se používá metoda melspectrogram z knihovny Librosa, která jako vstup dostane parametry n_fft(délka časové rámce pro výpočet FFT), y (pole časových rámců), hop_length(určuje překrytí časových rámců), n_mels(určuje počet pásem Mel frekvencí). Jako výstup dostáváme dvourozměrné pole, počet pásem Mel frekvencí * počet časových rámců. Dále se v metodě použijí tato data na vykreslení grafu, který se uloží do příslušného adresáře. Pro výpočet MFCC se používá metoda mffc, která dostává parametry y (pole časových rámců), sr (rozsah vzorkovací frekvence), n_mfcc (počet MFFC dílků) a dct_type (typ cosinovi diskrétní funkce) jako vstup. Výstupem je sekvence MFCC dílků v čase. Dále se v metodě kontroluje, jestli jsme opravdu dostali požadovaný tvar, případně se doopraví. V další části se vygeneruje graf pro zobrazení diagramu a uloží se. K tomu se

využívá knihovny Matplotlib (MATPLOTLIB DEVELOPMENT TEAM, 2018) a funkce specshow (LIBROSA DEVELOPMENT TEAM, 2019). Jako výstup funkce load_track se vrací dvě hodnoty. První je výsledné pole z převodu na spektrogram, případně MFCC a druhá hodnota značí poměr mezi vstupním tvarem a vzorkovací frekvencí. Na obrázku 26 je zobrazena metoda load_track pro MFFC. Nakonec se všechny data uloží do definovaného souboru.

```
def load_track(filename, enforce_shape=None):
    new_input, sample_rate = lbr.load(filename, duration=30, mono=True)
    features = lbr.feature.mfcc(y=new_input, sr=sample_rate, n_mfcc=40, dct_type=2)

    if enforce_shape is not None:
        if features.shape[0] < enforce_shape[0]:
            delta_shape = (enforce_shape[0] - features.shape[0],
                    enforce_shape[1])
            features = np.append(features, np.zeros(delta_shape), axis=0)
        elif features.shape[0] > enforce_shape[0]:
            features = features[: enforce_shape[0], :]

        if features.shape[1] < enforce_shape[1]:
            delta_shape = (enforce_shape[0],
                           enforce_shape[1]- features.shape[1])
            features = np.append(features, np.zeros(delta_shape), axis=1)
        elif features.shape[1] > enforce_shape[1]:
            features = features[: enforce_shape[1], :]

        plt.figure(figsize=(10, 4))
        specshow(features, fmax = 8000,x_axis = 'time')
        plt.colorbar(format='%+2.0f')
        plt.title('MFCC')
        plt.tight_layout()
        plt.savefig('mfccfinal\\%s' % os.path.basename(filename)[:-2] + 'png', format='png')
        plt.close()

    features[features == 0] = 1e-6
    return (np.log(features), float(new_input.shape[0]) / sample_rate)
```

Obrázek 26 - metoda load_track (vlastní zpracování)

6.2.2 Stavba a trénování modelu

Stavba a trénování modelu probíhá v souboru train-model.py. Skript obsahuje parametry:

- -d, --dataset_path, kde se definují vstupní data modelu.
- -m, --model_path, kde se definuje výstupní model.

Hlavní funkce souboru jsou build_model a train_model. Funkce build_model má jako parametr vstupní rozměry dat. Budování modelu probíhá pomocí vrstev z Keras knihovny. Konkrétně pomocí vrstev: Convolution2D, MaxPooling2D, Lambda, Bidirectional, Dense, Flaten, Concatenate a Input.

- Covolution2D – Slouží k vytvoření konvoluční vrstvy pro dvojrozměrné vstupy. Vrstva má mnoho parametrů, mezi hlavní patří: filters – určuje kolik má konvoluční vrstva filtrů, kernel_size – určuje velikost jádra, strides – určuje posun jádra, activation – určuje typ aktivační funkce.
- MaxPooling2D - Vrstva pro sdružování dat. Hlavní parametry vrstvy jsou: pool_size – určuje velikost sdružovacího okna, srides – určuje posun okna.
- Lambda – Vrstva, která pomáhá měnit strukturu dat. Má tyto parametry: function – funkce která mění struktura dat, out_shape – očekávaný tvar výstupu po provedení parametru function.
- Bidirectional – Vrstva umožňující rekurentním vrstvám stát se obousměrnými. Parametry: layer – určuje typ Rekurentní vrstvy, merge_mode – určuje, jak budou kombinovány výstupy.
- Dense – Vrstva umožňující propojit s předchozí vrstvou všechny neurony. Parametry má units – počet neuronů ve vrstvě, activation – určuje druh aktivační funkce
- Flaten – Vrstva která zploští vstup do jedné dimenze
- Concatenate – Spojuje dva vstupy do jedné vrstvy
- Input – Používá se jako první vrstva v modelu a definuje, jak budou vypadat vstupní data

Model vytváří pomocí vytvoření instance třídy Model s parametry vstupní a výstupní vrstvy. Funkce vrací instanci modelu. Na obrázku 27 je zobrazena funkce build_model, kde můžeme vidět, jak je vybudován model.

```
def build_model(input_shape):
    input_layer = Input(input_shape, name='input')
    layer = input_layer
    layer = Convolution2D(filters=16, kernel_size=3, strides=1, padding='valid', data_format="channels_last", activation='relu')(layer)
    layer = MaxPooling2D(pool_size=2, strides=2)(layer)
    layer = Convolution2D(filters=32, kernel_size=3, strides=1, padding='valid', data_format="channels_last", activation='relu')(layer)
    layer = BatchNormalization(momentum=0.02)(layer)
    layer = MaxPooling2D(pool_size=2, strides=2)(layer)

    layer = Convolution2D(filters=64, kernel_size=3, strides=1, padding='valid', data_format="channels_last", activation='relu')(layer)
    layer = MaxPooling2D(pool_size=2, strides=2)(layer)
    layer = Convolution2D(filters=32, kernel_size=3, strides=1, padding='valid', data_format="channels_last", activation='relu')(layer)
    layer = MaxPooling2D(pool_size=4, strides=4)(layer)

    layerBi = input_layer
    layerBi = MaxPooling2D(pool_size=3, strides=3)(layerBi)
    layerBi = Lambda(lambda x: x[:, :, :, 0])(layerBi)
    layerBi = Bidirectional(LSTM(16, return_sequences=True))(layerBi)
    layerBi = TimeDistributed(Dense(len(GENRES)))(layerBi)
    time_distributed_merge_layer = Lambda(
        function=lambda x: K.mean(x, axis=1),
        output_shape=lambda shape: (shape[0],) + shape[2:],
        name='output_merged'
    )
    layerBi = time_distributed_merge_layer(layerBi)
    layerBi = Dense(64)(layerBi)

    layer = Flatten()(layer)
    layer = Dense(256, activation='relu')(layer)

    layer = Concatenate()([layer, layerBi])
    layer = Dense(10, activation='softmax')(layer)
    model_output = layer
    model = Model(input_layer, model_output)
    return model
```

Obrázek 27 - funkce build_model (vlastní zpracování)

Funkce train_model dostává jako vstupní parametry tréninková data a cestu, kam bude uložen výsledný model. Pomocí funkce train_test_split z knihovny sklearn jsou data náhodně rozdělena na dvě množiny. Prvá množina je tréninková a druhá je testovací. Parametr test_size funkce train_test_split určuje poměr velikosti množin. Hodnota 0.1 znamená, že 90 procent dat jsou tréninková a 10 procent testovací. Po získání modelu z funkce build_model je model kompilován funkcí compile. Kde se nastavují parametry pro trénování. V programu jsou nastaveny tyto parametry:

- loss – určuje, jaká metoda bude použita jako chybová funkce
- optimizer – udává metodu pro optimalizování modelu.
- metricts – určuje, jaké veličiny budou zobrazovány během učení

Metoda fit třídy Model provádí trénování modelu. Její vstupní parametry jsou:

- x – určuje množinu tréninkových dat
- y – určuje množinu správných žánrů, kde data v množině x na pozici indexu i, značí žánr dat v množině y s indexem i
- batch_size – určuje velikost dat pro průběžné testování
- epochs – určuje počet cyklů trénování modelu

- validation_data – určuje množinu testovacích dat, která se spustí po každém cyklu
- verbose – určuje způsob výpisu průběhu trénování
- callbacks – pole instancí třídy CallBacks, které slouží k logování a změně parametrů trénování. V našem modelu využíváme třídy Tensorboard, ModelCheckpoint a ReduceLROnPlateau. Tensorboard nám umožnuje grafické pozorování průběžných výsledků po každém cyklu. ModelCheckpoint ukládá model s nejvyšší úspěšností. ReduceLROnPlateau definuje, jak hodně se můžou měnit parametry modelu při optimalizaci.

Na obrázku 28 je zobrazena funkce train_model se všemi parametry.

```
def train_model(data, model_path):
    x = data['x']
    y = data['y']

    (x_train, x_val, y_train, y_val) = train_test_split(x, y, test_size=0.1,
            random_state=SEED)

    print('Building model...')
    x_train =np.expand_dims(x_train, axis=3)
    x_val = np.expand_dims(x_val, axis=3)

    n_features = x_train.shape[2]
    row = x_train.shape[1]
    input_shape = (row, n_features, INIT_CHANNEL)

    model = build_model(input_shape)
    opt = rmsprop()
    model.compile(
            loss='categorical_crossentropy',
            optimizer=opt,
            metrics=['accuracy']
        )

    print('Training...')
    tbCallBack = TensorBoard(log_dir='./Graph-final', histogram_freq=1, write_graph=True, write_images=True)
    model.fit(
        x=x_train, y=y_train, batch_size=BATCH_SIZE, epochs=EPOCH_COUNT,
        validation_data=(x_val, y_val), verbose=1, callbacks=[
            ModelCheckpoint(
                model_path, save_best_only=True, monitor='val_acc', verbose=1
            ),
            ReduceLROnPlateau(
                monitor='val_acc', factor=0.5, patience=8, min_delta=0.01, min_lr=0.0000000015,
                verbose=1
            ),
            tbCallBack
        ]
    )

    return model
```

Obrázek 28 - funkce train_model (vlastní zpracování)

Pro sledování hodnot a jednotlivé částích modelu při trénování, je spuštěn program Tensorboard. V něm je vidět stavba modelu, úspěšnost po jednotlivých cyklech trénování. Jakých hodnot nabývají jednotlivé části vrstev modelu. Program se spustí v konzoli příkazem: tensorboard --logdir ./Graph-final, kde parametr logdir je cesta k adresáři, kam se ukládají průběžné výsledky. Přístup je z prohlížeče na adrese localhost:6006 (SARKIS, 2017).

6.2.3 Aplikace modelu

O používání vygenerovaného modelu se stará skript v souboru tagging_music_data.py. Vstupní parametry skriptu jsou následující:

- -s, --songs_path – určuje cestu k adresáři písniček, u kterých se má zjistit žánr.
- -m, --model_path – definuje cestu k vytrénovanému modelu
- -o, --output_path – cesta k excelovému souboru kam budou zapsány výsledky tagování pro jednotlivé písničky

Hlavní metoda skriptu je predict_genre, která má vstupní parametry shodné se vstupními parametry skriptu. Z definovaného adresáře získá cesty všech písniček. Pak pro každou písničku volá metodu predict na objektu vytrénované neuronové sítě, jako parametr má hudební data, která byla načtena pomocí metody load_track. Výstup z metody predict je pole o rozměru počtu všech žánrů pro kterou byla síť trénována. Kde je hodnota v poli největší implikuje zvolení žánru pro danou písničku. Následně se vybere správný index vítězného žánru a k němu odpovídající popis. Nakonec se výsledek uloží do excelu. Pro zápis do excelového souboru typu xlsx je využívána knihovnu openyxl. Na obrázku 29 je zobrazení metody predict_genre.

```
def predict_genre(model_path, songs_path, output_path):

    model = load_model(model_path)

    wb = Workbook()
    sheet = wb.active

    songs = [f for f in listdir(songs_path) if isfile(join(songs_path, f))]
    for index, song in enumerate(songs):
        song_data = load_track(os.path.join(songs_path, song))[0]

        song_data = np.expand_dims(song_data, axis=3)
        song_data = np.expand_dims(song_data, axis=0)
        result = model.predict(song_data)
        selected_genre = GENRES[np.argmax(result)]
        sheet.cell(row=index + 1, column=1).value = songs[index]
        sheet.cell(row=index +1, column=2).value = "-"
        sheet.cell(row=index +1, column=3).value = selected_genre

    wb.save(output path)
```

Obrázek 29 - metoda predict_genre (vlastní zpracování)

6.3 Instalace

Pro nainstalování projektu je třeba mít nainstalovaný Python 3.5, pro který jsou nastaveny příslušné verze ostatních knihoven. Dále je zapotřebí mít nainstalovaný program Virtualenv, kde se vytvoří prostředí pro projekt a pomocí programu pip se nainstalují potřebné knihovny ze souboru requirements.txt. Dále už se můžou spouštět python skripty. Na obrázku 30 je seznam kroků potřebných k instalaci, vytvoření modelu a otagování písniček. Předpokládá se, že uživatel má naklonovaný projekt z Githubu(RYCHNA, 2019) nebo z DVD, má stažená tréninková data (KOZAKOWSKI a MICHALAK, 2016) ve složce data a vytvořené virtuální prostředí pomocí Virtualenv. V kořenové složce se spustí následující příkazy. Všechny příkazy mají nastavené defaultní adresáře. Pro změnu adresářů nebo názvu modelu je třeba specifikovat pomocí parametrů skriptů.

```
pip install -r requirements.txt
python create_data_pickle.py
python train_model.py
python tagging_music_data.py
```

Obrázek 30 - kroky instalace (vlastní zpracování)

7 Závěr a doporučení

Cílem diplomové práce bylo vytvořit model z neuronových sítí, který by byl schopný otagovat hudební písničky podle žánru. Podařilo se vytvořit model, který na datové množině GTZAN dosahoval úspěšnosti 75 procent. Pro transformaci hudebních dat byly zkoušeny dvě metody s dvěma nastaveními parametrů na stejném modelu. Používaly se metody MFCC a spektrogram s měnícím se parametrem počtu Mel pásem (128 a 256). Model s nejvyšší úspěšností používal metodu spektrogram a 256 Mel pásmy. Postavení neuronové sítě bylo inspirováno článkem Music Genre Classification with Paralleling Recurrent Convolutional Neural Network (FENG, LIU a YAO, 2017), bohužel se nepodařilo i při podobném nastavení sítě dosáhnout stejné úspěšnosti jako v dané práci. Největším problémem bylo přidání Embedding vrstvy doprostřed Bi-LSTM bloku, protože framework Keras umožňuje přidání vrstvy Embedding pouze jako první vrstvu modelu. Z článku, ze kterého se vycházelo je Embedding vrstva uprostřed modelu. Což nejspíš způsobuje o 15 procent nižší úspěšnost než, které bylo dosaženo v článku.

Další faktor, který by dokázal zlepšit kvalitu neuronové sítě, by bylo trénování na větší množině dat. Tím by se určitě dosáhlo daleko větší přesnosti, když budou tak obrovská data kvalitní. Ale na zpracování velkých objemů dat je potřeba dostatečně velký výpočetní výkon, což jsme v rámci práce neměli. Výpočty probíhaly na notebooku s procesorem Intel Core i7 -3537U při frekvenci 2.89GHz. Což není pro výpočty neuronových sítí moc vhodné. Lepší by bylo provádět výpočty na serverech nebo alespoň na notebooku s grafickou kartou s podporou pro Tensorflow.

Další oblastí, kterou by bylo dobré přidat, je uživatelské rozhraní. Nyní se všechno spouští jakou python skripty a k tomu, aby to mohl využít obyčejný uživatel, to není vhodné.

Nicméně, jako základ pro další vývoj modelu neuronové sítě posloužila tato práce velice dobře. V dalším vývoji bude pokračováno s využitím výkonnější výpočetní techniky a větší velikosti hudebních dat.

8 Literatura

LE, James. *The 10 Algorithms Machine Learning Engineers Need to Know* [online]. 2016 [cit. 2019-04-03]. Dostupné z: https://www.kdnuggets.com/2016/08/10-algorithms-machine-learning-engineers.html

BERKA, Petr. *Rozhodovací stromy* [online]. 2005 [cit. 2019-04-03]. Dostupné z: https://sorry.vse.cz/~berka/docs/izi456/kap_5.1.pdf

MEDALOVÁ, Kristína. Neuron a jeho stavba [online]. 2015 [cit. 2019-03-30]. Dostupné z: https://www.mentem.cz/blog/neuron/

HASSANKASHI, Mahsa. *Neural Network* [online]. 2019 [cit. 2019-04-03]. Dostupné z: https://www.codeproject.com/Articles/1200392/Neural-Network

SHARMA, Sager. Activation Functions: Neural Networks [online]. 2017 [cit. 2018-12-30]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

POLAMURI, Saimadhu. DIFFERENCE BETWEEN SOFTMAX FUNCTION AND SIGMOID FUNCTION [online]. 2017 [cit. 2018-12-30]. Dostupné z: http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/

SHEKHAR, Amit. What Is Bias In Artificial Neural Network? [online]. 2018, 10. 3. 2018 [cit. 2018-11-25]. Dostupné z: https://letslearnai.com/2018/03/10/what-is-bias-in-artificial-neural-network.html

GUPTA, Tushar. Deep Learning: Back Propagation [online]. 2017, 25. 1. 2017 [cit. 2018-11-25]. Dostupné z: https://towardsdatascience.com/back-propagation-414ec0043d7

MALADKAR, Kishan. 6 Types of Artificial Neural Networks Currently Being Used in Machine Learning. Analytics India Magazine [online]. 15. 1. 2018 [cit. 2018-10-26]. Dostupné z: https://www.analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/

GUPTA, Tushar. Deep Learning: Feedforward Neural Network. Towards Data Science [online]. 2017, 1. 5. 2017 [cit. 2018-10-26]. Dostupné z: https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7

CHANDRADEVAN, Ramraj. Radial Basis Functions Neural Networks—All we need to know [online]. 2017 [cit. 2018-11-03]. Dostupné z: https://towardsdatascience.com/radial-basis-functions-neural-networks-all-we-need-to-know-9a88cc053448

INSTITUT BIOSTATISTIKY A ANALÝZ MASARYKOVY UNIVERZITY. Jednoduchá samoorganizační mapa - organizační a aktivní dynamika. *Matematická biologie* [online]. 2018 [cit. 2019-04-03]. Dostupné z: https://is.muni.cz/www/98951/41610771/43823411/43823458/Analyza_a_hodnoc/44563149/07Neuronove_site_-_soutezive_site_.pdf

VOJÁČEK, Antonín. Samoučící se neuronová síť - SOM, Kohonenovy mapy [online]. 2006, 14 Květen, 2006 [cit. 2018-11-24]. Dostupné z: https://www.kiv.zcu.cz/studies/predmety/uir/NS/Samouc_NN2.pdf

DONGES, Niklas. *Recurrent Neural Networks and LSTM* [online]. 2018 [cit. 2019-04-03]. Dostupné z: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5

GANDHI, Rohith. Introduction to Sequence Models—RNN, Bidirectional RNN, LSTM, GRU [online]. 2018, 26. 6. 2018 [cit. 2018-11-25]. Dostupné z: https://yiranyyu.github.io/doc/Introduction%20to%20Sequence%20Models%20%E2%80%94%20RNN,%20Bidirectional%20RNN,%20LSTM,%20GRU.pdf

LI, Shuai, Wanqing LI, Chris COOK, Ce ZHU a Yanbo GAO. Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN [online]. 2018 [cit. 2018-12-30]. Dostupné z: https://www.researchgate.net/publication/323735015_Independently_Recurrent_Neural_Network_IndRNN_Building_A_Longer_and_Deeper_RNN

LI, Shuai. Development of Recurrent Neural Networks and Its Applications to Activity Recognition [online]. University of Wollongong, 2018 [cit. 2018-12-30]. Dostupné z: https://ro.uow.edu.au/cgi/viewcontent.cgi?article=1393&context=theses1

KARN, Ujjwal. An Intuitive Explanation of Convolutional Neural Networks [online]. 2018, 11.8.2016 [cit. 2018-11-25]. Dostupné z: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

HASSANKASHI, Mahsa. Machine Learning - Opinion and Text Mining by Naive Bayes Classifier [online]. 2016, 2018 [cit. 2018-11-25]. Dostupné z: https://www.codeproject.com/Articles/1197656/Machine-Learning-Opinion-and-Text-Mining-by-Naive

BERKA, Petr. Bayesovská klasifikace [online]. 2005 [cit. 2018-11-25]. Dostupné z: https://sorry.vse.cz/~berka/docs/izi456/kap_5.6.pdf

BERKA, Petr. *Statistika* [online]. 2005 [cit. 2019-04-05]. Dostupné z: https://sorry.vse.cz/~berka/docs/izi456/kap_3.pdf

SAYAD, Saed. *Linear Discriminant Analysis* [online]. 2018 [cit. 2019-04-05]. Dostupné z: http://www.saedsayad.com/lda.htm

BERKA, Petr. *Rozhodovací pravidla* [online]. 2005 [cit. 2019-04-05]. Dostupné z: https://sorry.vse.cz/~berka/docs/izi456/kap_5.3.pdf

HOLČÍK, Jiří a Martin KOMENDA. Úvod do genetických algoritmů [online]. 2015 [cit. 2018-11-25]. Dostupné z: http://portal.matematickabiologie.cz/res/f/uvod-do-generickych-algoritmu.pdf

LEBEN, Jakob. *Data Sets* [online]. 2015 [cit. 2019-04-05]. Dostupné z: http://marsyas.info/downloads/datasets.html

MAUTNER, Pavel. *Analýza signálů ve frekvenční oblasti: Fourierova transformace* [online]. 2014 [cit. 2019-04-08]. Dostupné z: http://www.kiv.zcu.cz/~mautner/Azs/Azs5_Fourierova_transformace.pdf

SMITH, Steven W. The Scientist and Engineer's Guide to Digital Signal Processing: Chapter 12: The Fast Fourier Transform [online]. 2003 [cit. 2018-11-25]. Dostupné z: http://www.dspguide.com/ch12/2.htm

PRAHALLAD, Kishore. Speech Technology: A Practical Introduction: Topic: Spectrogram, Cepstrum and Mel-Frequency Analysis [online]. 2001 [cit. 2018-11-25]. Dostupné z: http://www.speech.cs.cmu.edu/15-492/slides/03_mfcc.pdf

PERQLÄ, Pasi. *Mel-frequency cepstral coefficients (MFCCs) and gammatone filter banks* [online]. 2015 [cit. 2019-04-15]. Dostupné z: http://www.cs.tut.fi/~sgn14006/PDF2015/S04-MFCC.pdf

LYONS, James. Mel Frequency Cepstral Coefficient (MFCC) tutorial [online]. 2012 [cit. 2018-12-30]. Dostupné z: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

KURSA, Miron B., Witold RUDNICKI, Alicja WIECZORKOWSKA, Elżbieta KUBERA a Agnieszka KUBIK-KOMAR. Musical Instruments in Random Forest [online]. 2009 [cit. 2018-11-25]. Dostupné z: https://www.researchgate.net/publication/225207301_Musical_Instruments_in_Random_Forest

SILLA JR, Carlos N., Alessandro L. KOERICH a Celso A. A. KAESTNER. A Machine Learning Approach to Automatic Music Genre Classification [online]. 2008 [cit. 2018-11-25]. Dostupné z: https://link.springer.com/content/pdf/10.1007%2FBF03192561.pdf

XU Changsheng, Namunu C. MADDAGE, Xi SHAO, Fang CAO a Qi TIAN. Musical genre classification using support vector machines [online]. 2003 [cit. 2018-11-25]. Dostupné z: https://www.researchgate.net/publication/4015150_Musical_genre_classification_using_support_vector_machines

DONG, Mingwen. Convolutional Neural Network Achieves Human-level Accuracy in Music Genre Classification [online]. 27.2.2018 [cit. 2018-12-30]. Dostupné z: https://arxiv.org/pdf/1802.09697.pdf

IRVIN, Jeremy, Elliott CHARTOCK a Nadav HOLLANDER. Recurrent Neural Networks with Attention for Genre Classification [online]. 2012 [cit. 2018-12-30]. Dostupné z: http://cs229.stanford.edu/proj2016/report/IrvinChartockHollander-RecurrentNeuralNetworkswithAttentionforGenreClassification-report.pdf

FENG, Lin, Shenlan LIU a Jianing YAO. Music Genre Classification with Paralleling Recurrent Convolutional Neural Network [online]. 201ž [cit. 2018-12-30]. Dostupné z: https://arxiv.org/pdf/1712.08370.pdf

CHOI, Keunwoo, Gyorgy FAZEKAS, Mark SANDLER a Kyunghyun CHO. CONVOLUTIONAL RECURRENT NEURAL NETWORKS FOR MUSIC CLASSIFICATION [online]. [cit. 2019-01-06]. Dostupné z: https://arxiv.org/pdf/1609.04243.pdf

WU, Wenli, Guangxiao SONG, Zhijie WANG a Fang HAN. *Music Genre Classification Using Independent Recurrent Neural Network* [online]. 2018 [cit. 2019-04-08]. Dostupné z: http://www.cacpaper.com/files/CAC2018/141/141-Wu.pdf

LIANG, James. An Introduction to Deep Learning. *Towards Data Science* [online]. 2018 [cit. 2019-04-08]. Dostupné z: https://towardsdatascience.com/an-introduction-to-deep-learning-af63448c122c

CHANGHAU, Isaac. *Loss Functions in Neural Networks* [online]. 2018 [cit. 2019-04-15]. Dostupné z: https://isaacchanghau.github.io/post/loss_functions/

SKALSKI, Piotr. *How to train Neural Network faster with optimizers?* [online]. 2018 [cit. 2019-04-15]. Dostupné z: https://towardsdatascience.com/how-to-train-neural-network-faster-with-optimizers-d297730b3713

KOZAKOWSKI, Piotr a Bartosz MICHALAK. *Music Genre Recognition* [online]. 2016 [cit. 2019-04-15]. Dostupné z: http://deepsound.io/music_genre_recognition.html

TENSORFLOW [online]. 2019 [cit. 2019-04-15]. Dostupné z: https://www.tensorflow.org/

TensorBoard: Visualizing Learning [online]. 2019 [cit. 2019-04-15]. Dostupné z: https://www.tensorflow.org/guide/summaries_and_tensorboard

PYTHON SOFTWARE FOUNDATION. *Python* [online]. 2019 [cit. 2019-04-15]. Dostupné z: https://www.python.org/

PYTHON SOFTWARE FOUNDATION. *Pip* [online]. 2019 [cit. 2019-04-19]. Dostupné z: https://pypi.org/project/pip/

CHACON, Scott a Jason LONG. Git. [online]. 2019 [cit. 2019-04-19]. Dostupné z: https://git-scm.com/

GITHUB COMPANY, Scott. *GitHub* [online]. 2019 [cit. 2019-04-19]. Dostupné z: https://github.com/

LIBROSA DEVELOPMENT TEAM. *Librosa* [online]. 2018 [cit. 2019-04-15]. Dostupné z: https://librosa.github.io/librosa/

MATPLOTLIB DEVELOPMENT TEAM. *Matplotlib* [online]. 2018 [cit. 2019-04-15]. Dostupné z: https://matplotlib.org/

BICKING, Ian. *Virtualenv* [online]. 2018 [cit. 2019-04-19]. Dostupné z: https://virtualenv.pypa.io/en/latest/

SARKIS, Anthony. *Tensorboard quick start in 5 minutes.* [online]. 2017 [cit. 2019-04-15]. Dostupné z: https://medium.com/@anthony_sarkis/tensorboard-quick-start-in-5-minutes-e3ec69f673af

RYCHNA, Lukáš. *Music Tagging* [online]. 2019 [cit. 2019-04-25]. Dostupné z: https://github.com/rychnal/music_tagging

Seznam příloh

1. Dvd s programem, textem práce a vlastními obrázky