Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Hluboké neuronové sítě a jejich použití

Bc. Jakub Sadílek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jakub Sadílek

Informatika

Název práce

Hluboké neuronové sítě a jejich použití

Název anglicky

Deep neural networks and their usage

Cíle práce

Cílem této diplomové práce je aplikace neuronových sítí prostřednictvím frameworku TensorFlow a Keras na řešení úlohy za využití dat dostupných volně na internetu. Dílčím cílem bude rozbor samotné problematiky hlubokých neuronových sítí, oblastí jejich aplikace a nástrojů pro jejich realizaci.

Metodika

Po prostudování teoretických poznatků zoblasti hlubokých neuronových sítí bude sestavena rešerše na dané téma. Rešerše zohlední odvětví a úlohy, v kterých jsou hluboké neuronové sítě typicky aplikovány. Zahrne i rozbor nejpoužívanějších nástrojů pro realizaci hlubokých neuronových sítí se zaměřením zejména na framework TensorFlow a Keras.

Praktická část práce se zaměří na řešení úlohy z oblasti analýzy a klasifikace zvukových záznamů. Pro tuto úlohu budou využita data z datové sady AudioSet – velkorozměrné sady ručně anotovaných zvukových záznamů od firmy Google. Neuronová síť bude realizována pomocí frameworku TensorFlow a Keras. Vstupní zvuky bude klasifikovat na základě analýzy jejich spektrogramu. Výsledkem práce bude ověření funkčnosti navrženého řešení na zvolených datech.

Doporučený rozsah práce

60

Klíčová slova

deep learning, rozpoznávání zvuků

Doporučené zdroje informací

GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Beijing ; Boston ; Farnham ; Sevastopol ; Tokyo: O'Reilly, 2019. ISBN 978-1-4920-3264-9.

CHOLLET, F. Deep learning v jazyku Python, Grada, Praha, 2019, ISBN 978-80-247-3100-1

Předběžný termín obhajoby

2022/23 ZS – PEF

Vedoucí práce

doc. Ing. Arnošt Veselý, CSc.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 31. 10. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 30. 03. 2024

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Hluboké neuronové sítě a jejich použití" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31.3.2024 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu mé diplomové práce doc. Ing. Arnoštu Veselému, CSc. za jeho užitečné rady a připomínky, které mi při psaní práce pomohly.

Hluboké neuronové sítě a jejich použití

Abstrakt

Tato diplomová práce zkoumá možnost využití specificky vybrané podmnožiny velkorozměrné datové sady audio záznamů AudioSet pro trénování hluboké neuronové sítě řešící úlohu klasifikace zvukových událostí. V rámci teoretické části práce je sestavena rešerše na téma neuronových sítí, hlubokého učení a metodik a nástrojů aplikovatelných v procesu trénování hlubokých neuronových sítí. V praktické části je následně popsáno, jakým způsobem byla vybrána podmnožina kategorií zvukových událostí z celého datasetu, jakým způsobem byla získána použitelná data a jakým způsobem byla data transformována pro účely trénování hluboké neuronové sítě. Nakonec je popsán iterativní proces trénování neuronové sítě, v němž jsou aplikovány poznatky z teoretické části. V závěru práce jsou vyhodnoceny výsledky trénování a použitelnost zvoleného datasetu pro danou úlohu.

Klíčová slova: hluboké neuronové sítě, hluboké učení, audio klasifikace, AudioSet, TensorFlow, Keras, Python

Deep neural networks and their usage

Abstract

This thesis explores the possibility of utilizing a specifically selected subset of a large-scale audio dataset AudioSet for training a deep neural network to solve the task of sound event classification. Theoretical part of the thesis explores the topics of neural networks, deep learning, and methodologies and tools applicable in the process of training deep neural networks. The practical part then describes how a subset of sound event classes was selected from the full dataset, how usable data was obtained, and how the data was transformed for the purposes of training the deep neural network. Finally, the iterative process of neural network training is described, in which findings from the theoretical part are applied. The conclusion of the work evaluates the training results and the usability of the chosen dataset for the given task.

Keywords: deep neural networks, deep learning, audio classification, AudioSet, TensorFlow, Keras, Python

Obsah

1 Úvod

Obor umělé inteligence zahrnující strojového učení, hluboké učení, neuronové sítě a další odvětví zažívá v posledních letech stále rychlejší vzestup, umožněný technologickým rozvojem počítačového hardwaru a rozšiřováním uživatelské báze internetu. Díky stále většímu propojení lidí a firem po celém světě je stále snazší získávat potřebná data na nichž se neuronové sítě jakožto nejprověřenější nástroj strojového učení dokáží trénovat a přinášet uživatelům stále pokročilejší funkce. Tato data zahrnují informace o uživatelích internetu nebo informace extrahované z obsahu, který stále rychleji rostoucí báze uživatelů internetu tvoří. S minimalizací osobní elektroniky, příchodem chytrých telefonů a nástupem sociálních sítí a technologických gigantů byl růst množství obsahu uloženého na internetu ještě více akcelerován. Tento obsah nabývá všech forem od textového obsahu po multimediální audio a video. Když jsou tato data zakódovaná v obsahu internetu správně vytříděna a popsána mohou vzniknout užitečné datové sady jakou je například i AudioSet.

AudioSet je velkorozměrnou datovou sadou volně dostupnou na internetu popisující segmenty audio stop extrahovatelných z veřejně přístupné databáze uživatelského video obsahu YouTube. Tato práce se zaměřuje na využití AudioSetu pro trénování hluboké neuronové sítě, pro klasifikaci zvukových jevů do několika definovaných kategorií. Vybrané kategorie ze zmíněného datasetu budou patřit do množiny zvuků znamenajících ohrožení osob nebo majetku a neuronová síť bude sloužit jako prostředník pro jejich rozpoznání.

Cílem práce bude ověřit je-li využití vybraného datasetu pro trénování hluboké neuronové sítě možné v omezených technologických podmínkách, které nedovolují využít celý dataset ale pouze jeho specifický subset na míru dané úloze. Hluboká neuronová síť bude vytvořena pomocí frameworku TensorFlow a knihovny Keras. Tyto knihovny budou představeny v teoretické části práce, ve které bude zároveň nastíněn vývoj strojového učení a neuronových sítí a bubou rozebrány metodiky trénování neuronových sítí. Teoretická část práce poskytne důležitá informační východiska pro následné zpracování praktické části.

2 Cíl práce a metodika

Cílem této diplomové práce je aplikace neuronových sítí prostřednictvím frameworku *TensorFlow* a *Keras* na řešení úlohy za využití dat dostupných volně na internetu. Dílčím cílem bude rozbor samotné problematiky hlubokých neuronových sítí, oblastí jejich aplikace a nástrojů pro jejich realizaci.

Po prostudování teoretických poznatků z oblasti hlubokých neuronových sítí bude sestavena rešerše na dané téma. Rešerše zohlední odvětví a úlohy, v kterých jsou hluboké neuronové sítě typicky aplikovány. Zahrne i rozbor nejpoužívanějších nástrojů pro realizaci hlubokých neuronových sítí se zaměřením zejména na framework *TensorFlow* a *Keras*.

Praktická část práce se zaměří na řešení úlohy z oblasti analýzy a klasifikace zvukových záznamů. Pro tuto úlohu budou využita data z datové sady AudioSet – velkorozměrné sady ručně anotovaných zvukových záznamů od firmy *Google*. Neuronová síť bude realizována pomocí frameworku *TensorFlow* a *Keras*. Vstupní zvuky bude klasifikovat na základě analýzy jejich spektrogramu. Výsledkem práce bude ověření funkčnosti navrženého řešení na zvolených datech.

3 Teoretická východiska

3.1 Strojové učení

Strojové učení představuje podmnožinu oblasti umělé inteligence, která se zásadně odlišuje od tradičního programování. Umělá inteligence na svém počátku v padesátých letech 20. století pracovala s myšlenkou manuálního programování expertního systému, který následně byl schopný provádět rozhodnutí na základě vnitřních parametrů, které mu byly pevně vloženy. Toto paradigma bylo dostačující pro svou dobu a setrvalo v povědomí inženýrů do počátku devadesátých let 20. století. Toto paradigma nese název symbolická umělá inteligence. (Chollet, [2018])

Jiný přístupu k vývoji umělé inteligence razilo odvětví strojového učení a koncept umělých neuronů. Tento obor byl ve skutečnosti prozkoumáván už v roce 1943, kdy byl představen první zjednodušený matematický model toho, jak pravděpodobně může fungovat komunikace mezi neurony v mozcích živých organismů. Úspěch umělých neuronů a neuronových sítí předpovídal světlou budoucnost inteligentních strojů, avšak technologické limitace tento obor následně zamrazily až do opětovného oživení v osmdesátých letech 20. století. (Géron, [2017])

Narozdíl od symbolické umělé inteligence, která vyžaduje explicitní instrukční sadu pro každou operaci, systémy strojového učení se samy učí z poskytnutých dat. Tento proces trénování zahrnuje vystavení systému velkému množství příkladů, které jsou relevantní pro úkol, který má systém řešit. Celý koncept je zjednodušeně převráceným obrazem klasického programování, kde je počítači předložen program a data a na základě těchto vstupů jsou vyvozovány výsledky. U strojového učení jsou vstupem data a výsledky a počítač na základě vstupů vyvozuje pravidla.

Vezměme si jako příklad úkol automatického označování fotografií. V tradičním programování by bylo nutné definovat pevná pravidla pro identifikaci objektů na fotografiích, to může být náročné a neefektivní, jelikož takový typ dat nemá pevnou deterministickou strukturu. Strojové učení však tento proces převrací. Místo definování pravidel systém dostane velké množství fotografií, které už byly dopředu označeny. Tyto fotografie slouží jako tréninkový soubor, který umožňuje systému naučit se rozpoznávat vzory a vztahy v datech.

Během tréninku systém analyzuje fotografie, hledá statistické struktury a vzory, které mohou být spojeny s konkrétními štítky. Postupně se tímto způsobem systém učí identifikovat a klasifikovat různé prvky na obrázcích, jako jsou lidé, objekty nebo scény, a přiřazuje jim příslušné štítky. Například, pokud je většina fotografií s modrým nebem označena jako "pláž", systém se postupně naučí spojovat modré nebe s tímto štítkem. (Chollet, [2018])

3.2 Neuronové sítě

Neuronové sítě jsou základním stavebním kamenem strojového učení a umělé inteligence. Jejich vznik inspirovala architektura lidského mozku, respektive biologické neurony a synapse mezi nimi. Někteří vědci a výzkumníci v oboru umělé inteligence argumentují, že by se od této biologické analogie mělo ustoupit, neboť obor umělých logických neuronů a neuronových sítí se – co se fungování týče – vydává zcela jiným směrem než jeho biologický protějšek a může tak ovlivňovat naši představivost, kam až obor umělých neuronových sítí může růst a vyvíjet se. Tento argument může být v mnoha ohledech validní, neuronové sítě se vskutku neustále vzdalují svému biologickému protějšku, avšak držení se této analogie nemusí být problém nýbrž užitečná pomůcka, dovolíme-li si vydávat se za hranice toho co funguje v biologickém systému. (Géron, [2017])

Historie neuronových sítí sahá až do roku 1943, kdy neurofyziolog Warren McCulloch a matematik Walter Pitts představili zjednodušený matematický model fungování neuronů v mozku živých organismů. (Géron, [2017]) Objev těchto vědců byl následován hypotézou neuroplasticity Hebbem a objev perceptronu Rosenblatem (Haykin, 2009), avšak technologická limitace tehdejší doby nedovolila oboru pokračovat v rapidním rozvoji až do osmdesátých let 20. století, kdy došlo k opětovnému oživení tohoto odvětví. K restartu oboru přispěly vynálezy nových metod učení a architektur neuronových sítí. (Géron, [2017])

V devadesátých letech vytlačila umělé neuronové sítě z výsluní metoda podpůrných vektorů (SVM), která přinášela dostačující výsledky se solidním základem v oblasti teorie. (Géron, [2017]) Návrat k neuronovým sítím, nastal následně v roce 2010, kdy došlo k průlomovým pokrokům pod záštitou odborných týmů z několika světových univerzit. Tehdy popularita neuronových sítí opět začala v odborné komunitě růst a tentokrát se už nezastavila. V roce 2011 začaly být neuronové sítě úspěšně aplikovány na úlohy

rozpoznávání obrázků, které do té doby představovaly komplikovaný problém. V daném roce byla díky neuronovým sítím trénovaným na grafických kartách tato úloha vyřešena s přesností 74,3 %, následující rok to bylo již 83,6 % a v roce 2015 byla tato úloha uzavřena jako vyřešená s úspěšností 96,4 %. Tento trend zajistil neuronovým sítím opět místo na vrcholu zájmu odborníků a jistou aplikaci v dalších oborech. V poslední dekádě (2010-2020) přinesly umělé neuronové sítě technologickou revoluci, která dosáhla dříve nedosažitelných cílů od klasifikace obrazů, porozumění psanému textu a řeči na téměř lidské úrovni po vylepšení algoritmů pro překlad jazyka, digitální asistenty, zaměřování reklam nebo hraní her. (Chollet, [2018])

Rozvoji neuronových sítí přispělo několik faktorů:

1. Velké a stále rychleji rostoucí množství dat, se kterými mohou neuronové sítě pracovat a učit se z nich. Tato data plynou z digitalizace rozličných aspektů našeho světa, kterou jsme v posledních dekádách zažili a zapojení většího a většího množství lidí do online aktivit. (Géron, [2017])
2. Významný nárůst výpočetního výkonu, díky miniaturizaci a v důsledku Mooreova zákona, který tvrdí, že počet tranzistorů v procesorech poháněících naše počítače se každé dva roky zdvojnásobuje. (Moore, 1965) Nemalý vliv měl taky rozmach herního průmyslu, který dal za vznik trhu produkujícího výkonné grafické karty, na kterých lze umělé neuronové sítě trénovat v praktickém časovém rámci. (Géron, [2017]) Tento technologický vývoj dále posílil snahy výzkumníků, kteří nově své nápady mohli rychleji iterovat a uvádět do praxe. Vývoj neuronových sítí, strojového učení a hlubokého učení se tak lépe ukázal spíše jako praktická disciplína, kde lze hypotézy v mnoha případech ověřovat empiricky spíše než teoreticky. (Chollet, [2018])
3. Vylepšení trénovacích algoritmů, byť změny v algoritmech nebyly nijak velké, měly velký pozitivní vliv na jejich efektivitu. (Géron, [2017])
4. Některé teoretické limitace, které původně vyvolávaly pochybnosti o efektivitě neuronových sítí byly vyvráceny. Významným příkladem je obava, že trénovací algoritmy uvíznou v lokálních optimech a nedokáží se tak efektivně učit. Tento problém byl však v praxi z velké části vyvrácen a bylo dokázáno že buď jen zřídka skončí na lokálních optimech, nebo jsou tato optima dostatečně blízká globálnímu optimu, aby byla užitečná.

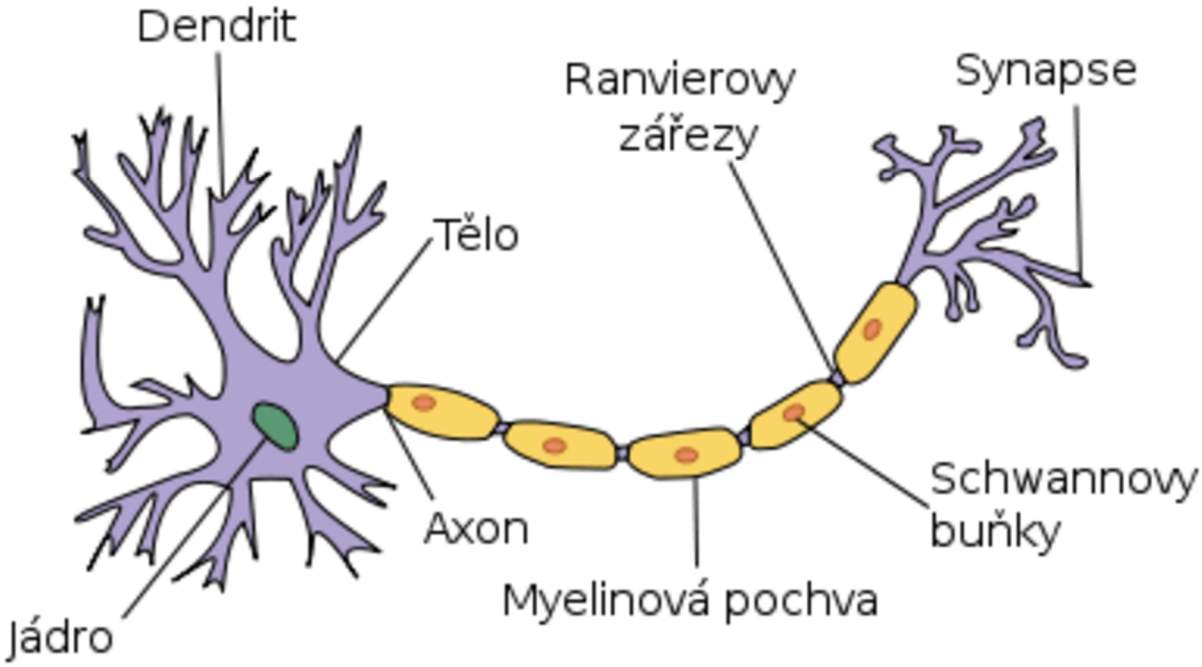
5. Konečně se zdá, že neuronové sítě a tím pádem umělá inteligence obecně vstoupily do sebe posilující křivky růstu zájmu, vývoje a financování. Postupně vstoupily v patrnost i u veřejnosti a investorů, kteří pomáhají proces vývoje stále více urychlovat a vytvářet nové průlomové aplikace.(Géron, [2017])

3.3 Neuron

Neuron je základním stavebním prvkem umělé neuronové sítě i biologického mozku. Pro pochopení funkce umělého neuronu je vhodné představení jeho biologického protějšku, neboť oba typy neuronů sdílí některé základní vlastnosti.

Biologický neuron se skládá z těla buňky, které obsahuje buněčné jádro a tvoří většinu jeho objemu. Z těla se větví krátké výrůstky zvané dendrity a jeden nápadně dlouhý (jeho délka se může pohybovat od několikanásobku délky těla až po tisícinásobky) zvaný axon. Axon je na konci rozvětven v kolaterály, které jsou zakončeny synapsemi. Synapse jsou připojeny na dendrity nebo přímo na tělo jiných neuronů.

Biologické neurony fungují tak, že skrze ně putují krátké elektrické signály z jiných neuronů prostřednictvím synapsí. Potom co neuron obdrží dostatečné množství impulzů, odešle vlastní impulz do dalších neuronů, na které je navázán svým axonem. (Géron, [2017])



Obrázek 1 - Neuron – Zdroj: (Neuron, 2006)

3.4 Umělý neuron

Bylo zmíněno, že myšlenka umělých neuronových sítí a umělého neuronu vychází ze snahy McCullocha a Pittse zjednodušeně matematicky modelovat fungování biologických neuronů v mozkových tkáních živých organismů při snaze pochopit jejich funkci. Umělý neuron je základním stavebním kamenem neuronové sítě stejně jako je biologický neuron základním stavebním kamenem mozkové tkáně.

Hlavním společným prvkem biologického i umělého neuronu je fakt, že jsou svou funkcí samy od sebe velmi jednoduché, avšak ve spojení s mnoha dalšími neurony projeví na základě jevu emergence složitější chování. Složitost tohoto chování je potom závislá na architektuře propojení jednotlivých neuronů. (Géron, [2017])

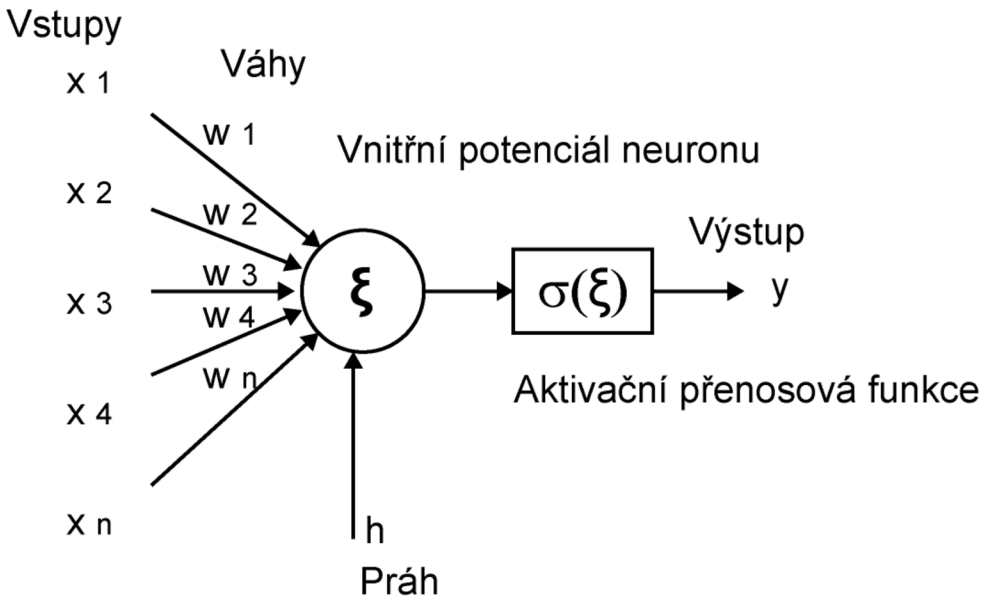
Neurony jsou schopné akumulovat a využívat zkušenosti procesem učení neboli procesem při němž jsou z okolí získávány informace, které může následně neuronová síť používat jako svou znalostní bázi. To je jedna z nejzásadnějších podobností umělých neuronových sítí a mozku. Druhou zásadní podobností je metoda, jakou jsou tyto informace ukládány a později využívány. Uvnitř neuronové sítě dochází při procesu učení k změnám sil vazeb mezi jednotlivými neurony, které lze také pojmenovat jako synaptické váhy. (Haykin, 2009)

Umělý neuron má následující části, kterých si lze povšimnout na obrázku č. 2:

- Vstupy na obrázku reprezentované jako x_n představující synapse umělého neuronu. Na vstupech neuron přijímá vstupní hodnoty, s kterými následně pracuje.
- Váhy nebo synaptické váhy. Každý vstup má svou váhu, která u umělého neuronu na rozdíl od biologického neuronu může nabývat hodnot od -1 do 1. Každá hodnota vstupující do neuronu je příslušnou vahou vynásobena. Váhy propůjčují neuronům v neuronové síti schopnost ukládat informace tím, že jsou průběžně při procesu učení modifikovány na takovou konfiguraci vah, při níž neuronová síť vykazuje požadované chování. (Haykin, 2009)
- Sumarizační funkce pro sečtení všech vstupních hodnot vynásobených svými váhami. Výstupem této funkce je vnitřní potenciál neuronu.
- Práh je volitelný vstupní parametr sumarizační funkce, který lze neuronu předložit pro ovlivnění jejího výstupu. (Géron, [2017])

- Aktivační funkce limituje sílu výstupu neuronu. Na vstupu přijímá vnitřní potenciál neuronu a jeho hodnotu omezí v rámci svého definičního oboru.

Schéma umělého neuronu je vyobrazeno na následujícím obrázku:



Obrázek 2 - Umělý neuron - Zdroj:(Holčík, 2015)

Uvnitř umělého neuronu se potom co přijme na vstupech vstupní hodnoty, v zásadě provádějí dvě hlavní operace, a to výpočet sumarizační a aktivační funkce.

1. Sumarizační funkce provede vážený součet vstupních hodnot. Tuto operaci lze vyjádřit matematickým zápisem na příkladu 1:

$$v = \sum_{j=1}^{m} w_j x_j \qquad (1)$$

A následně k váženému součtu přičte volitelný práh (h) čímž získá vnitřní potenciál neuronu (příklad 2).

$$\xi = v + h \qquad (2)$$

2. Aktivační funkce (σ) vypočte výstup neuronu (y) přenesením vnitřního potenciálu neuronu (ξ) do svého oboru hodnot (příklad 3).

$$y = \sigma(\xi) \tag{3}$$

Aktivační funkce mohou být schodové nebo sigmoidní. Více o aktivačních funkcích v následující kapitole.

(Haykin, 2009) (Holčík, 2015)

3.5 Aktivační funkce

Jak již bylo zmíněno aktivační funkce je součástí umělého neuronu, která limituje amplitudu jeho výstupu. Funkce přijímá vnitřní potenciál neuronu jako svou vstupní hodnotu, kterou následně omezí na základě svého definičního oboru v konečném rozmezí. Tato funkce je klíčovou součástí umělého neuronu při převodu vstupních signálů na výstupní hodnoty. Aktivační funkce v zásadě určuje, jak neuron reaguje na soubor vstupů. Tedy zda bude výstup aktivován nebo potlačen. Tato vlastnost je zásadní pro fungování umělých neuronových sítí.

Aktivační funkce hrají klíčovou roli v učení neuronových sítí. Během procesu učení se váhy mezi neurony upravují na základě chyb ve výstupu sítě. Aktivační funkce zde pomáhá určit, jak se mají tyto váhy měnit, což je základem schopnosti neuronových sítí učit se.

Výběr vhodné aktivační funkce může mít velký dopad na výkon a efektivitu neuronové sítě. Například, některé funkce mohou vést k rychlejší konvergenci při učení nebo pomoci předcházet problémům, jako je vanishing gradient, kdy se aktualizace vah stávají příliš malými pro efektivní učení.

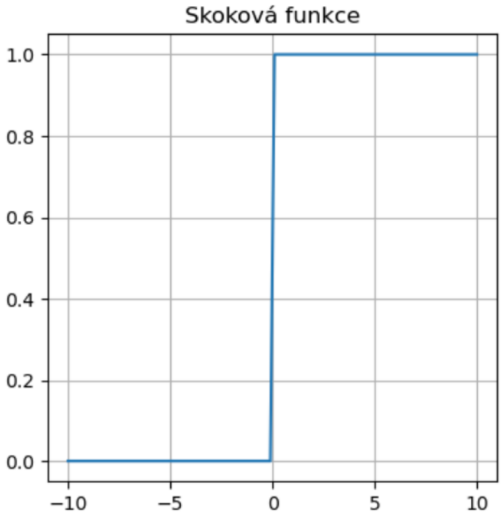
Existuje mnoho druhů aktivačních funkcí, které se používají v umělých neuronových sítích. Volba typu aktivační funkce je jedním z důležitých kroků při stavbě neuronové sítě, který podstatně ovlivňuje její fungování. (Sharma, 2020)

Některé základní typy aktivačních funkcí budou rozebrány v následujícím textu.

3.5.1 Prahová funkce

Prahová funkce je nejzákladnějším typem aktivační funkce, která se v umělých neuronových sítích používá. Definiční obor hodnot nabývá typicky pouze binárních hodnot 0 a 1. Přičemž nejčastěji nabývá hodnoty 1 pokud vstupní vnitřní potenciál neuronu nabývá kladné hodnoty. Lze ji tedy vyjádřit tímto předpisem (č. 4) a grafem (č. 1).

$$\sigma(\xi) = \begin{cases} 1 & pokud\ \xi \geq 0 \\ 0 & pokud\ \xi < 0 \end{cases} \tag{4}$$



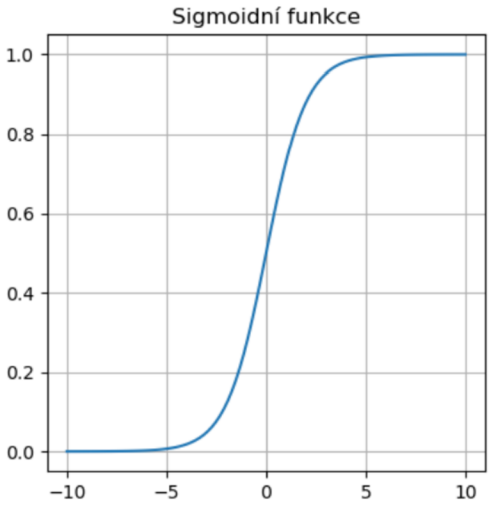
Graf 1 - Skoková funkce - Zdroj: Autor

Tato aktivační funkce byla použita v prvním modelu umělého neuronu představeného McCullochem a Pittsem v roce 1943.

3.5.2 Sigmoidní funkce

Ve sféře dopředných neuronových sítí je Sigmoidní funkce (někdy označovaná jako logistická) poměrně významná. Tato nelineární funkce, známá svými pravidelným tvarem ve tvaru písmene S, je hlavně používána ve výstupních vrstvách například pro úlohy binární klasifikace a logistické regrese. (Nwankpa, 2020)

Lze ji vyjádřit tímto grafem (č. 5).



(5)

Graf 2 - Sigmoidní funkce - Zdroj: Autor

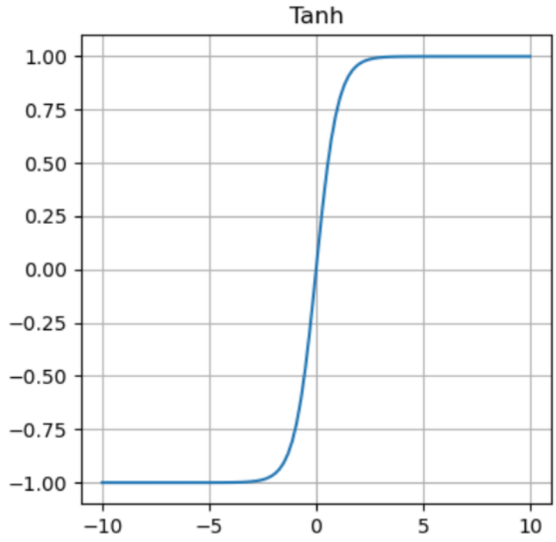
Použití sigmoidní funkce se však neobejde bez komplikací. Čelí například problému mizejícího gradientu při zpětné propagaci ze skrytých vrstev. Také je poměrně pomalá při učení a vede k nekonzistenci, protože její výstupy nejsou centrovány kolem nuly. Existují alternativy, které se těmto nedostatkům snaží vyhnout, jako například hyperbolický tangens. (Nwankpa, 2020)

3.5.3 Hyperbolický tangens

Hyperbolický tangens neboli Tanh funkce je důležitá aktivační funkce, která se sigmoidními aktivačními funkcemi sdílí hladkou S-křivku a rozmezí výstupů od -1 do 1. Narozdíl od sigmoidních funkcí je však Tanh nulově symetrická. Je preferována před sigmoidní funkcí v neuronových sítích s větším počtem vrstev kvůli lepším výsledkům učení. Tanh funkce se používá především v rekurentních neuronových sítích. (Sharma, 2020) (Nwankpa, 2020)

Předpis (č. 6) a graf (č. 3) Tanh funkce vypadá následovně:

$$\sigma(\xi) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6}$$



Graf 3 - Tanh funkce - Zdroj: Autor

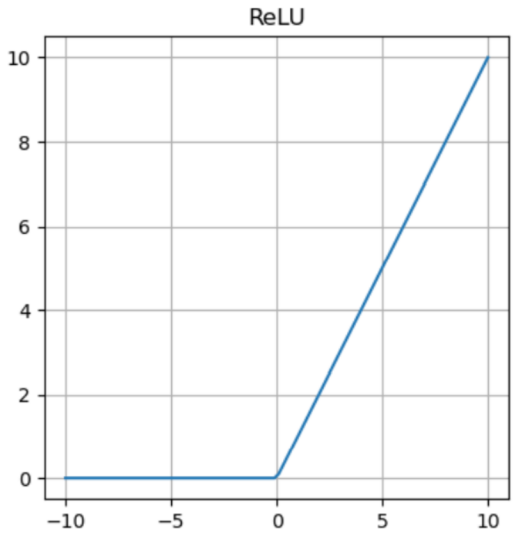
Nicméně Tanh funkce sdílí se sigmoidní funkcí problém mizejícího gradientu. Její hlavní výhodou je tedy nulově symetrický výstup, který pomáhá při procesu zpětné propagace. Unikátní rys Tanh funkce je, že dosahuje maximálního gradientu 1 pouze tehdy, je-li vstup roven nule. Tato vlastnost občas vede k problému tzv. mrtvých neuronů. Tento nedostatek vedl k vývoji dalších aktivačních funkcí, jako je například funkce ReLU. (Nwankpa, 2020) (Sharma, 2020)

3.5.4 **ReLU**

ReLU (Rectified Linear Unit) je nelineární aktivační funkce, která je v oboru neuronových sítích velmi hojně používaná a je preferovanou aktivační funkcí v oboru hlubokého učení a hlubokých neuronových sítí. Vyznačuje se svou rychlostí výpočtu což ve výsledku urychluje učící proces. Proto svou efektivitou předčí sigmoidní funkce i tanh.

ReLU funkce je definována tímto matematickým předpisem (č. 7) a grafem (č. 4).

$$f(x) = max(0, x) = \begin{cases} x_i & pokud\ x \geq 0 \\ 0 & pokud\ x < 0 \end{cases} \tag{7}$$



Graf 4 - Funkce ReLU - Zdroj: Autor

Své klíčové vlastnosti, tedy rychlosti výpočtu ReLU dosahuje tím, že se vyhýbá složitým matematickým operacím, tím že je v částečně lineární. Zároveň vrací nulový výstup pro záporné vstupy což ji činí imunní vůči problému mizejícího gradientu. Tím že tolik vstupů nuluje však vytváří neaktivní neurony, jejichž váhy nejsou při zpětné propagaci aktualizovány. Tento problém řeší některé další modifikované ReLU funkce (například leaky ReLU. Navíc je ReLU náchylnější k přeučení než sigmoidní funkce, tento problém je často řešen technikami jako dropout. (Sharma, 2020) (Nwankpa, 2020)

3.5.5 Softmax

Aktivační funkce Softmax je v podstatě kombinací několika sigmoidních funkcí. Každá sigmoidní funkce vrací hodnoty mezi 0 a 1, které můžeme vnímat jako pravděpodobnosti výskytu různých klasifikačních tříd. Na rozdíl od sigmoidních funkcí určených pro binární klasifikaci je Softmax funkce přizpůsobena pro úlohy klasifikace s vícero třídami.

Úlohy tohoto typu řeší díky tomu, že v modelech vytvořených pro klasifikaci s vícero třídami obsahuje výstupní vrstva sítě takový počet neuronů, který odpovídá počtu klasifikačních tříd. Softmax funkce na každém výstupním neuronu tak zajistí rozdělení výpočtu pravděpodobnosti výskytu na každém neuronu, tedy pro každou třídu. (Nwankpa, 2020) (Sharma, 2020)

Softmax funkci lze vyjádřit tímto matematickým předpisem (č. 8).

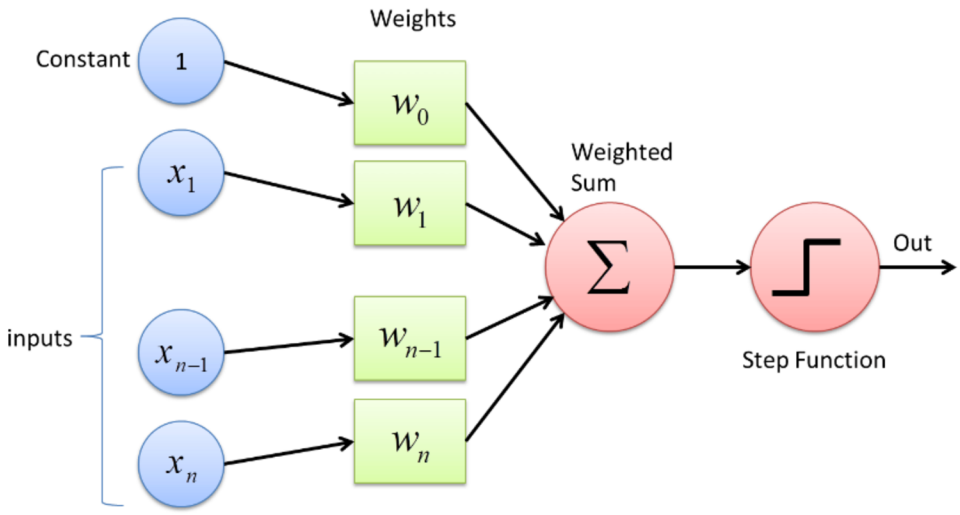
$$f(x) = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{8}$$

3.6 Perceptron

Perceptron představuje ranou formu architektury umělých neuronových sítí. Byl vytvořen Frankem Rosenblattem v roce 1957 a implementuje teoretické principy ustanovené McCullochem a Pittsem na modelu umělého neuronu. Model neuronu perceptronu má však upravenou formu zvanou lineární prahová jednotka (LTU Linear Treshold Unit). Na rozdíl od původního modelu pracuje LTU se vstupy a výstupy které nesou číselné hodnoty nikoli pouze binární hodnoty jako u původního modelu. Každý vstup je v něm vynásoben jeho specifickou váhou. Hlavní funkcí LTU je výpočet váženého součtu vstupů, na který je následně aplikována typicky prahová aktivační funkce, která určuje výstup perceptronu.

Struktura perceptronu je jednoduchá, skládá se z jedné LTU vrstvy, ve které je každý neuron propojen se všemi vstupy. Vstupy perceptronu jsou někdy zastoupeny speciálními vstupními neurony, které propouští data skrz do LTU vrstvy. Dodatečně v modelu figuruje doplňkový vstupní neuron s prahovým parametrem, který vždy nabývá hodnoty 1.

Na následujícím obrázku (č. 3) je vyobrazeno schéma perceptronu:

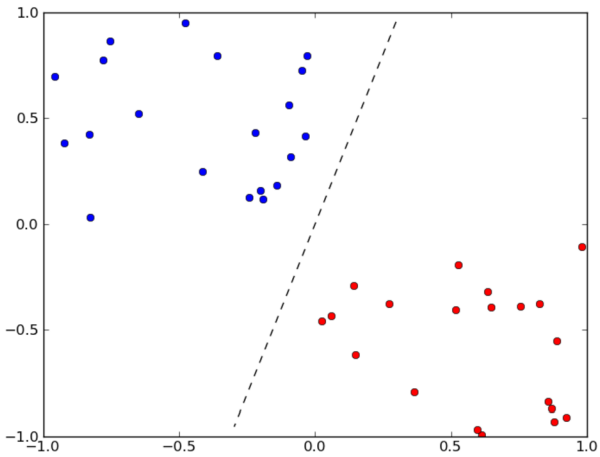


Obrázek 3 - Perceptron - Zdroj:(Chowdhury, 2023)

Architektura perceptronu umožňuje klasifikaci vstupních dat do odlišných kategorií na základě lineárních kombinací vstupů. Když vypočítaný součet překročí určitý práh, perceptron vrátí kladný výstup. V opačném případě vrací výstup negativní.

Proces učení perceptronu je inspirován Hebbovým zákonem učení. Ten tvrdí, že neurony, které se aktivují společně, posilují svá vazby. Algoritmus učení Perceptronu upravuje váhy mezi neurony na základě korelace jejich výstupů. Algoritmus v zásadě posiluje spojení, která vedou ke správným výstupům a oslabuje ty, která vedou k těm nesprávným. To se děje prezentací jednotlivých trénovacích instancí Perceptronu, který poté upravuje své váhy na základě přesnosti svého výstupu.

Schopnost perceptronu klasifikovat vstupní data do kategorií lze demonstrovat na v m-rozměrném prostoru. Tento prostor je naplněn různými vstupními hodnotami a perceptron se snaží tyto vstupy klasifikovat do předem definovaných tříd. Výstup perceptronu určuje zařazení vstupních dat do dané třídy pomocí rozdělení do oddělených rozhodovacích regionů.



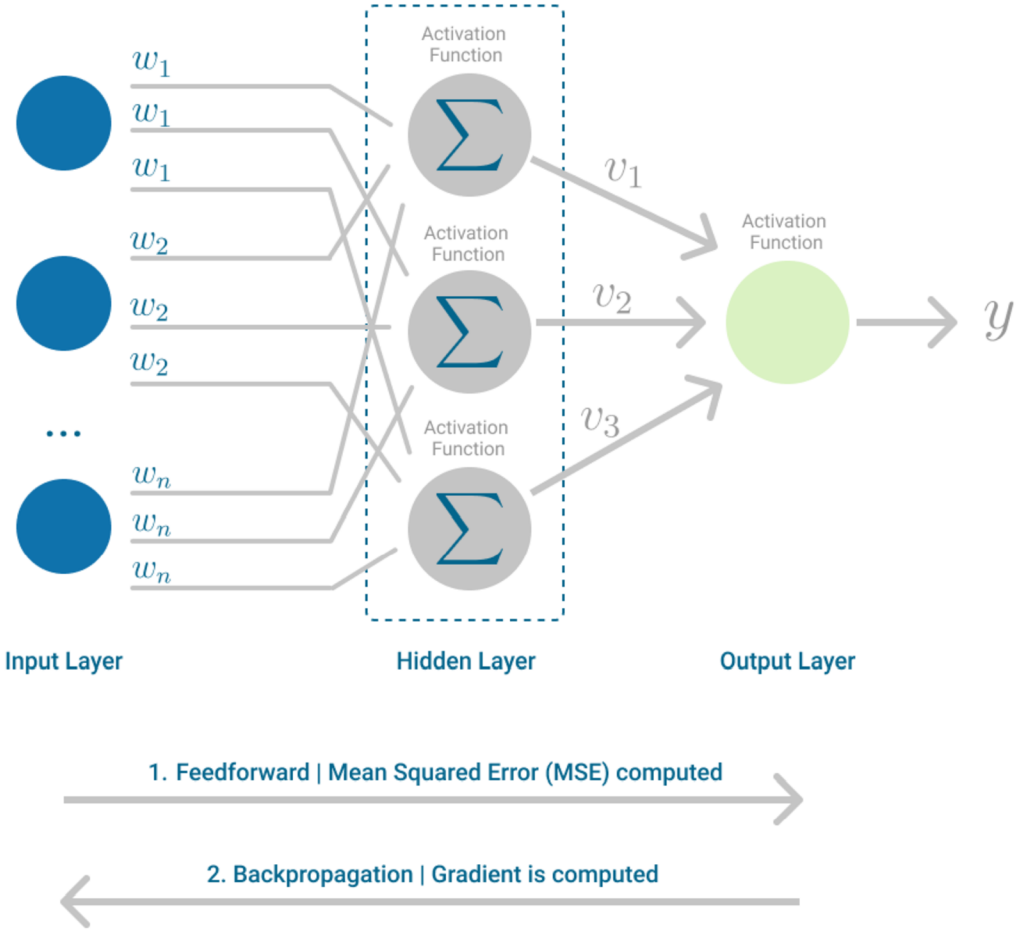
Graf 5 - Klasifikace perceptronu - Zdroj: (Sharma, 2017)

Schopnost perceptronu klasifikovat vstupy do několika tříd současně činí z něj všestranný více výstupový klasifikátor schopný zvládat složité klasifikační úlohy, přestože se na první pohled svou architekturou jeví jako zdánlivě jednoduchý. (Haykin, 2009) (Géron, [2017])

3.7 Vícevrstvý perceptron

Vícevrstvý perceptron je základní formou neuronové sítě rozšiřující základní principy jednovrstvého perceptronu popsaného v předchozí kapitole. Hlavním rozdílem je přítomnost více než jedné vrstvy neuronů. Vícevrstvý perceptron obsahuje vstupní vrstvu, jednu nebo více skrytých vrstev a výstupní vrstvu. Skryté vrstvy jsou klíčovým prvkem, který umožňuje vícevrstvému perceptronu učit se a reprezentovat mnohem složitější vzory a funkce, což je zásadní pro řešení nelineárních problémů.

Vícevrstvé perceptrony se liší od svého jednovrstvého předchůdce také v typu aktivačních funkcí. Zatímco jednovrstvý perceptron používá lineární prahové jednotky (LTU), vícevrstvé varianty často využívají nelineární aktivační funkce, jako jsou ReLU, sigmoidní funkce nebo hyperbolický tangens. Tyto nelineární funkce umožňují vícevrstvému perceptronu zachytit složitější vzorce ve vstupních datech.



Obrázek 4 - Vícevrstvý perceptron - Zdroj:(Bento, 2021)

Klíčovou technikou používanou v trénování vícevrstvých perceptronů je tzv. zpětná propagace (backpropagation). Proces zpětné propagace zahrnuje průchod vstupních dat sítí od vstupní vrstvy k výstupní, výpočet chybnosti výstupu a následnou úpravu vah neuronů na základě této chyby, a to v opačném směru, od výstupní vrstvy až ke vstupní. Tato metoda umožňuje efektivní učení i v komplexních neuronových sítích s mnoha vrstvami.

Díky své vícevrstvé struktuře a schopnosti modelovat nelineární vztahy jsou vícevrstvé perceptrony schopné řešit mnohem složitější úlohy než původní jednovrstvý model. Nacházejí uplatnění v různých oblastech strojového učení, včetně rozpoznávání obrazu, zpracování přirozeného jazyka a mnoha dalších aplikací. Tento zásadní pokrok v návrhu architektury neuronových sítí otevřel dveře novým disciplínám strojového učení jako je hluboké učení a odvětví hlubokých neuronových sítí. (Géron, [2017])

3.8 Hluboké učení

Hluboké učení napomáhá počítačovým systémům chápat složitá data, která v minulosti byla brána za strojově nezpracovatelná. Základem tohoto jevu je proces vytváření smysluplných reprezentací těchto dat. Reprezentacemi se rozumí rozličné způsoby, jak na data hledět. Pro vytvoření použitelné reprezentace dat je nutné je smysluplně transformovat do požadované podoby, s kterou se dá lépe pracovat. (Chollet, [2018])

Pro lepší ilustraci, například reprezentací zvukového záznamu může být jeho vlnový průběh neboli záznam jeho amplitudy v čase, jeho nejzákladnější reprezentace. Smysluplnější reprezentací zvukového záznamu pak může být například spektrogram nebo krátkodobá Fourierova transformace.

Na druhé straně pro obrazová data můžeme reprezentacemi chápat například mapy vlastností (tzv. feature maps), na kterých lze snadněji rozpoznat hrany, textury nebo specifické vzorce. Dále pak například histogramy barev nebo různé způsoby kódování barevného prostoru jako RGB, RGBA, HSL nebo CMYK.

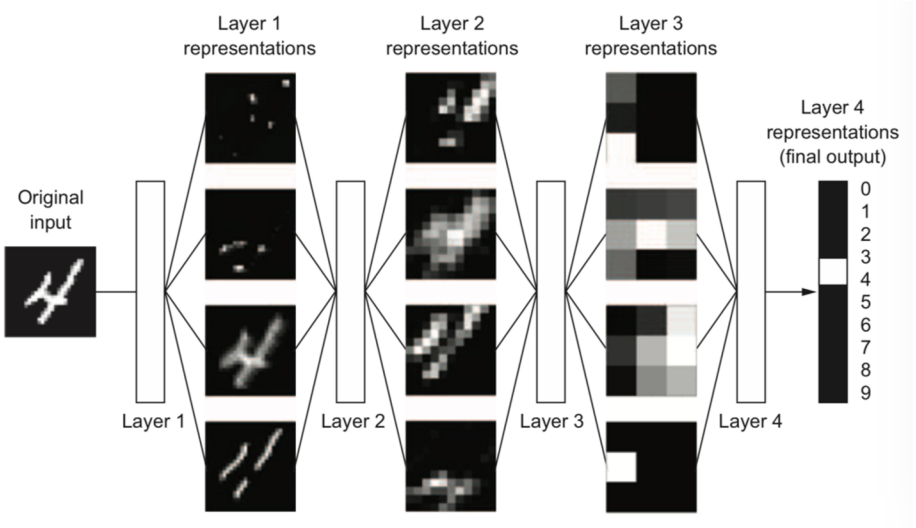
V konečném důsledku může být nejspecifičtější a nejsmysluplnější reprezentací vstupních dat jejich zařazení do nějaké definované třídy. Například zpracováváme-li úlohu detekce určitého objektu v obrazových datech, je pak finální námi požadovanou reprezentací pravděpodobnost zařazení vstupních dat do kategorie *obsahuje hledaný objekt* nebo

neobsahuje hledaný objekt. Nebo pokud řešíme úlohu klasifikace zvukových záznamů, bude nás například zajímat jaké hudební nástroje v daném záznamu zazněly, respektive s jakou pravděpodobností se záznamu dá přisoudit zařazení do jednotlivých tříd v klasifikační úloze. (Doshi, 2021)

Hluboké učení se od běžných metod strojového učení odlišuje jedním zásadním aspektem, a to sice schopností vytvářet reprezentace reprezentací, které jsou progresivně smysluplnější než ty předchozí. Toho hluboké neuronové sítě docilují svými skrytými vrstvami. V zásadě každá následující vrstva transformuje výstup předchozí vrstvy na smysluplnější reprezentaci vrstvy předchozí. Tento jednoduchý koncept v sobě skrývá velkou sílu. Hluboké učení postupně z nejzákladnějších reprezentací dat extrahuje smysluplnější a smysluplnější reprezentace, které lze postupně transformovat na konečný požadovaný výsledek, tedy konečnou reprezentaci dat pro řešenou úlohu. (Chollet, [2018])

Způsob, jakým hluboká neuronová síť zpracovává data lze také chápat jako postupnou extrakci užitečných informací pro danou úlohu, kde data prochází několika na sebe navazujícími filtry a z každého vyjdou očištěna o informace nadbytečné. (Chollet, [2018])

Proces transformace vstupních dat na smysluplnější reprezentace je ilustrován přehledně na následujícím diagramu. V tomto případě se jedná o základní úlohu rozpoznání psaných číslovek. Na vstupní vrstvě vidíme obrázek obsahující psanou číslici. V dalších skrytých vrstvách pak hluboká neuronová síť data transformuje na užitečnější reprezentace až po výstup, který určuje, o jakou číslovku se jedná.



Obrázek 5 - Reprezentace neuronové sítě - Zdroj:(Chollet, [2018])

Tento proces postupné extrakce užitečných vlastností vstupních dat také vede k dalšímu důvodu, proč hluboké učení má navrch nad běžnými metodami strojového učení. Hluboké neuronové sítě totiž tuto extrakci provádí zcela samostatně bez toho, aby programátor musel specificky definovat, jak má vybrané vlastnosti hledat. Hluboké neuronové sítě jsou zkrátka vystaveny velkému množství trénovacích dat a po každém průběhu učícího algoritmu si vytvářejí specifičtější a přesnější představu o tom jaký má být jejich požadovaný výstup. (Chollet, [2018])

K vytvoření vlastního modelu hlubokého učení lze zvolit dva přístupy popsané v následujících podkapitolách.

3.8.1 Učení nového modelu

Učení nového modelu hlubokého učení od základů je náročný proces, který vyžaduje shromáždění velkého množství dobře anotovaných dat a vytvoření architektury hluboké neuronové sítě tak aby byla schopná efektivně se učit požadované vlastnosti. Tento přístup je vhodný zejména pro řešení nových úloh, pro něž zatím neexistují funkční nebo dostatečně kvalitní modely.

Jednou z velkých výzev při trénování vlastního modelu od základu je potřeba shromáždit a anotovat dostatečně velkou datovou sadu. Tento proces může být časově náročný a nákladný, zejména v případech, kdy data mají specifickou charakteristiku pro danou úlohu. Tomuto úskalí lze jít naproti použitím již anotovaného datasetu z veřejné domény, avšak jen v případech, kdy taková data lze použít pro řešenou úlohu. Existence takových datasetů často implikuje i existenci už hotových modelů, které na nich byly trénovány. V případě, kdy takový model skutečně existuje se nabízí volit možnost takový model doučit na specifickém datasetu pro danou řešenou úlohu, který již může být menšího rozsahu.

Přestože trénování modelu hlubokého učení od základu je složité a časově náročné, jeho výhody jsou nezpochybnitelné hlavně v případech, kdy existující modely selhávají v řešení požadované úlohy nebo v případě, kdy řešení takové úlohy vůbec neexistuje. Modely trénované od základu mohou být účelně přizpůsobeny specifickým potřebám a charakteristikám dané úlohy, což často vede k efektivnějšímu řešením. (Parente, 2023)

3.8.2 Přenosové učení

Přenosové učení (tzv. transfer learning) je technika často aplikovaná v hlubokém učení, která zahrnuje přizpůsobení existujícího modelu původně trénovaného pro jednu úlohu ke zpracování jiné příbuzné úlohy.

Při použití přenosového učení model nezačíná od základu, ale staví na znalostech (extrahovaných vlastnostech nebo reprezentacích), které se již naučil v původním učícím procesu. Tato metoda může výrazně urychlit učební proces, neboť model potřebuje pouze upravit své vnitřní parametry pro specifickou úlohu a nemusí se tak učit vše od začátku.

Jednou z největších výzev v hlubokém učení je potřeba velkého množství dobře anotovaných dat. Přenosové učení tuto potřebu zmírňuje tím, že umožňuje modelu trénovanému na velkém datasetu aplikovat své naučené znalosti na novou úlohu, k čemuž stačí využít menší dataset. To je obzvláště užitečné u řešení úloh, pro něž je sběr dat nákladný nebo náročný.

Využitím předem natrénovaných modelů umožňuje aplikaci modelů hlubokého učení na širší škálu úloh, dokonce i těch, pro které dostupná data nejsou dostatečně kvalitní nebo obsáhlá k trénování modelu od základu. To rozšiřuje potenciální použití hlubokého učení v různých oborech, jako je zpracování přirozeného jazyka, počítačové vidění, porozumění snímaným medicínským datům a další.

Navíc trénování modelů hlubokého učení od nuly vyžaduje velký výpočetní výkon a čas. Přenosové učení snižuje tyto nároky využitím již existující struktury a vah předem natrénovaného modelu, čímž se stává použitelnější možností, zejména když jsou výpočetní zdroje omezené.

Výhodou procesu přenosového učení je to, že v závislosti na povaze úlohy a její podobností s původní úlohou pro níž byl navržen základní model mohou být různé vrstvy upraveného modelu zamrazeny nebo selektivně doladěny jen do té míry jakou vyžaduje nová úloha. Kompromis v rámci flexibility učení nového modelu proto u přenosového učení není tak velký, jak se může na první pohled jevit.

V souhrnu lze říci, že přenosové učení jako nástroj nabízí zejména efektivitu snižováním nároků na data a zdroje a zaručuje flexibilitu, čímž činí pokročilé aplikace hlubokého učení přístupnějšími napříč různými obory. (Sharma, 2023)(Brownlee, 2019a)

3.9 Učení neuronové sítě

Ve své základní podstatě strojové učení automatizuje proces tvorby analytických modelů. Místo explicitního programování systému s konkrétními instrukcemi pro každý možný scénář, neuronové sítě vytváří pravidla pro zpracování úloh samy, na základě poskytnutých vstupních dat. To činí strojové učení a neuronové sítě velmi cenným pro úlohy, kde je programování explicitních pravidel buď příliš složité nebo nerealizovatelné. Strojové učení je tak v zásadě opakem běžného programování analytických nebo expertních systémů, kde jsou naopak výstupem data a k jejich formování probíhá na základě pevně definovaných pravidel.

Vědomosti neuronových sítí jsou reprezentovány vahami spojení jednotlivých neuronů. Samotné učení neuronové sítě je tedy procesem, při němž dochází k úpravě hodnot těchto vah.

Na počátku učení jsou modelu neuronové sítě nastaveny prvotní váhy. Proces nastavení vah je komplexní a detailněji je popsán v kapitole Parametry, pro účely této kapitoly postačí zmínit, že váhy nabývají náhodných hodnot a po prvotním nastavení neuchovávají žádné užitečné vědomosti a síť tak není zatím schopna žádné užitečné predikce na základě vstupních dat.

Aby síť mohla predikci provádět je potřeba ji k tomu natrénovat. Trénovací proces je zjednodušeně procesem opakovaného vystavování sítě trénovacím datům. Na základě trénovacích dat síť provádí predikce, jejichž přesnost je porovnána s vzorovými výstupy a na základě rozdílu predikce a vzoru jsou aktualizovány váhy uvnitř sítě. Trénovací proces bude podrobněji rozebrán v následující kapitole. (Chollet, [2018])

3.10 Trénovací proces

Trénování neuronové sítě vyžaduje 3 základní věci:

1. **Vstupní data**

   Z těchto dat se bude neuronová síť učit. Data mohou mít různou podobu, například pro úlohu rozpoznání objektů v obrázku jsou vstupními daty samotné obrázky nebo pro úlohu klasifikace zvukových záznamů jsou vstupními daty zvukové soubory.
2. **Příklady očekávaných výstupů**

Neuronové síti je třeba předložit příklady požadovaných výstupů, aby mohla na základě odchylky správnosti svého výstupu od těchto příkladů definovat svou chybovost. Neuronová síť se následně učí minimalizovat svou chybovost.

3. **Způsob definice chybovosti výstupu**

   Způsob, jakým neuronová síť definuje svou chybovost je také důležité zvážit. Různé typy vstupních dat a výstupů vyžadují k evaluaci chybovosti odlišné přístupy a je proto důležité zvolit ten který pro daný typ vstupních dat a požadovaného výsledku dává největší smysl.(Chollet, [2018])

Samotné trénování neuronové sítě je procesem opakování několika základních kroků:

1. Výběr dávky

   Dávka obsahuje vstupních data, pro něž bude síť provádět predikce a očekávané výstupy, které poslouží jako reference pro kontrolu přesnosti výstupu sítě.

2. Dopředný průchod sítí

   Vstupní data z dávky jsou postupně umísťována na vstupy sítě, je proveden dopředný průchod sítí, během něhož síť na základě svých vah zpracuje vstupní data a na výstupu vrací predikce. Tyto predikce jsou výstupem aktuálního stavu sítě a její schopnosti zpracovat vstupní data.

3. Změření chyby

   Po dokončení dopředného průchodu a získání predikcí pro celou dávku dat je nutné vyhodnotit, jak si síť při predikci vedla. Toho lze docílit změřením rozdílu mezi predikcemi a očekávanými výstupy. Tento rozdíl neboli chyba, je klíčovým ukazatelem toho, s jakou přesností síť funguje.

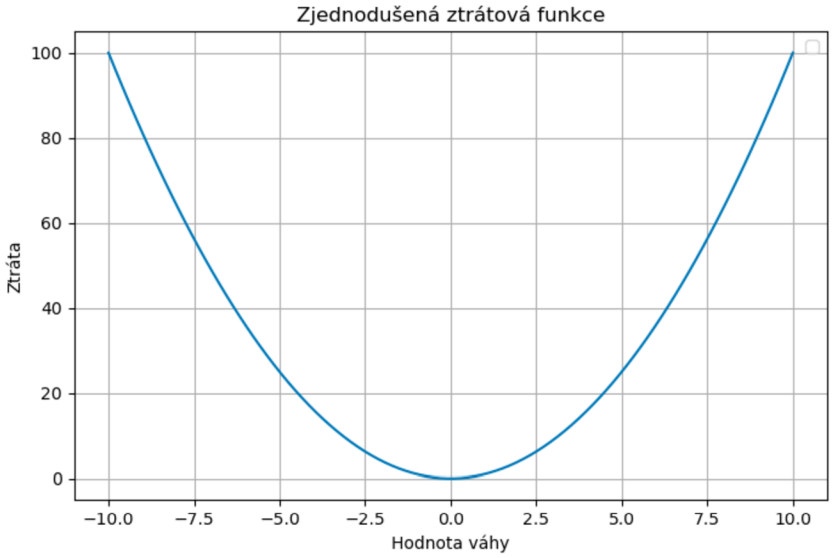
4. Aktualizace vah

   Provedení aktualizace vah v síti je založeno na metodě gradientního sestupu a metodě zpětné propagace chyby. Gradientní sestup hledá optimální nastavení vah tak, aby se minimalizovala celková chyba sítě, zatímco zpětná propagace zajišťuje efektivní šíření informací o chybě zpět skrze síť, aby došlo k aktualizaci vah směrem k nižší chybovosti sítě. (Chollet, [2018])

3.11 Metoda gradientního sestupu

Metoda gradientního sestupu je iterativní optimalizační algoritmus sloužící k nalezení lokálního minima vybrané spojité diferencovatelné konvexní funkce. Proces nalezení tohoto minima se nazývá konvergence.

Při trénování neuronové sítě potřebujeme najít pro každou váhu v síti takovou hodnotu, která povede k nejpřesnější predikci sítě neboli nejnižší ztrátě sítě. Zjištění ideální hodnoty každé váhy jednotlivě nezávisle na ostatních by však bylo extrémně složité a časově náročné. Zde přichází na řadu metoda gradientního sestupu, která pracuje s tzv. ztrátovou funkcí sítě. (Chollet, [2018])



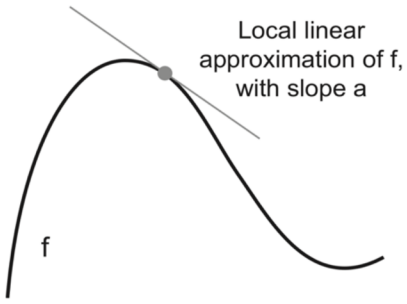
Graf 6 - Zjednodušená ztrátová funkce - Zdroj: Autor

Nejprve vysvětlení, co je ztrátová funkce. V nejjednodušší formě jde o funkci mapující hodnoty jedné váhy na ose X k hodnotám ztráty sítě na ose Y. Tato funkce je konvexní a diferencovatelná. Na grafu tuto funkci lze vyjádřit takto.

Z tohoto ilustrativního grafu je patrné, že k nejnižší ztrátě vede váha s hodnotou 0. Na počátku při inicializaci je hodnota váhy náhodně určena a cílem metody gradientní optimalizace je najít trasu, která od původní hodnoty povede k hodnotě ideální.

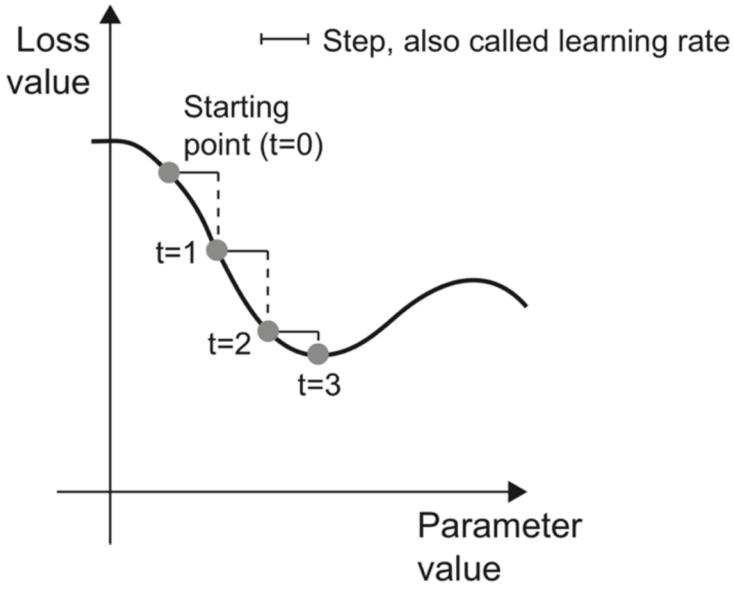
Aby bylo možné určit, že bylo takové ideální hodnoty váhy dosaženo a zároveň zjistit směr kterým se k ideální hodnotě vydat je potřeba v každém kroku optimalizace vypočítat derivaci funkce pro danou hodnotu váhy. Derivace funkce v tomto případě představuje sklon

funkce neboli její gradient. Ideální hodnota funkce se potom nutně musí nacházet v bodě, kde se sklon funkce blíží nule. (Roy, 2020)



Obrázek 6 - Derivace ztrátové funkce - Zdroj:(Chollet, [2018])

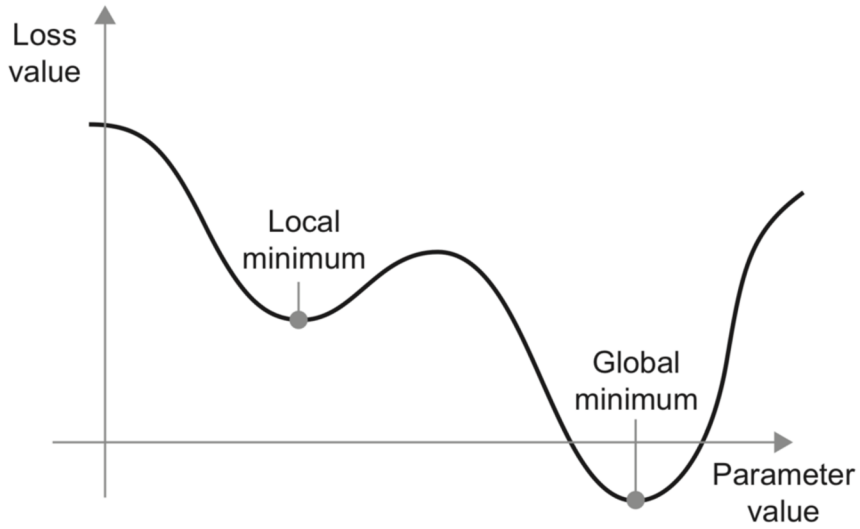
Hledání minima ztrátové funkce je prováděno po iteracích. V každé iteraci se metoda gradientního sestupu přiblíží minimu o jeden krok o definované velikosti. Velikost tohoto kroku neboli míra změny váhy v každé iteraci se také nazývá rychlost učení. Na nastavení hodnoty rychlosti učení závisí, jak dlouho neboli po kolik iterací, bude algoritmus k minimu směřovat. (Chollet, [2018])



Obrázek 7 - Rychlost učení - Zdroj:(Chollet, [2018])

Ztrátová funkce však má však v realitě složitější tvar obsahující více bodů, které optimalizační algoritmus může najít a chybně vyhodnotit jako správná řešení. Tyto body se

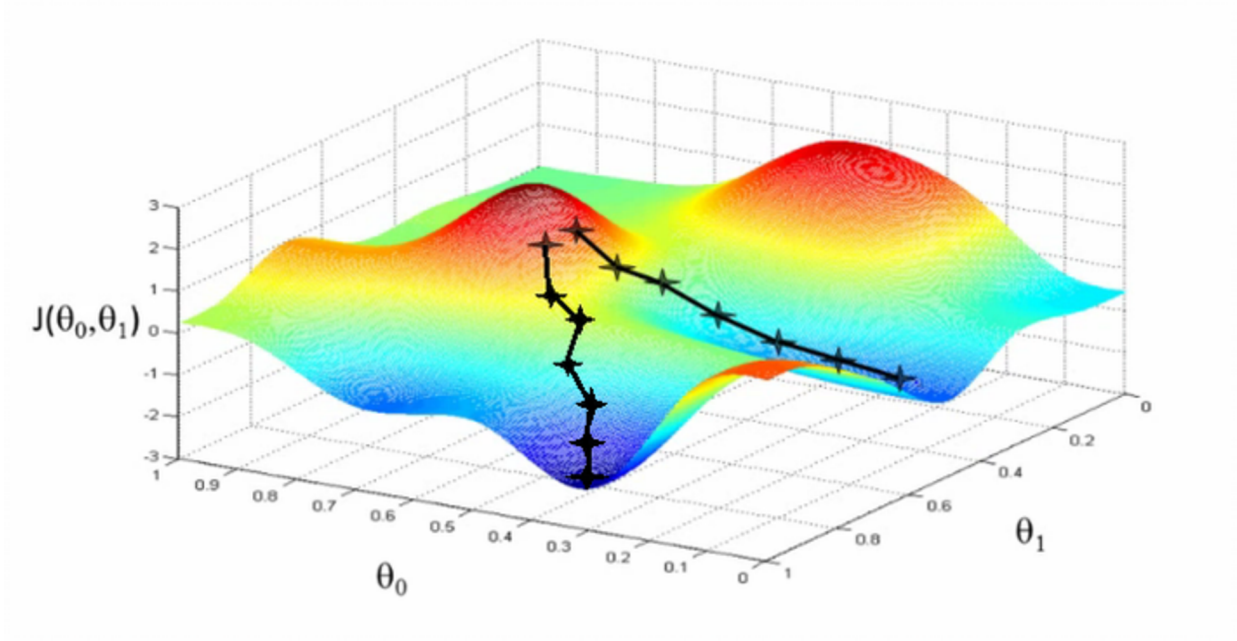
nazývají lokální minima. Skutečné ideální řešení, které se metoda gradientního sestupu snaží najít se nazývá globální minimum. Jedná se o bod s minimální hodnotou celé funkce. Schopnost algoritmu zaseknout se v lokálním minimu je také ovlivněna rychlostí učení. Čím je rychlost učení nižší, tím je pravděpodobnost nalezení lokálního minima větší. Tomuto problému lze předejít zvýšením rychlosti učení, což však není ideální řešení. V takovém případě může naopak nastat situace, kdy algoritmus zcela přeskočí globální minimum. Lepším přístupem je využití hyperparametru momentum, který algoritmu propůjčuje vlastnost setrvačnosti. Více o nastavení rychlosti učení a hyperparametru momentum v kapitole Hyperparametry.



Obrázek 8 - Globální a lokální minima - Zdroj:(Chollet, [2018])

Doposud byla ztrátová funkce, jejíž minimum metoda gradientního sestupu hledá, popsána pouze v jednom rozměru, respektive na dvourozměrném grafu, kde jeden rozměr vždy reprezentuje ztrátovou funkci. To je však velmi silné zjednodušení. Metoda gradientního sestupu hledá optimální hodnoty všech trénovatelných parametrů modelu současně, právě v tom je její síla. Z toho důvodu má gradient ztrátové funkce právě tolik rozměrů, kolik je trénovatelných parametrů jejichž hodnoty algoritmus hledá. Tento n-dimenzionální prostor nelze rozumně zobrazit a popsat v běžném 3D vnímání světa, natož v jeho 2D reprezentaci. Proto se často pro zjednodušení zobrazuje gradient ve třech rozměrech, pro dva trénovatelné parametry a hodnotu ztráty. Tento gradient má podobu zvlněného terénu, obsahujícího různě vysoké vrcholy, planiny a různě hluboká údolí neboli lokální minima a globální minimum. Pro ještě větší intuitivnost procesu gradientního sestupu

se často uvádí příměr s míčkem, který se z náhodného bodu v tomto terénu musí dostat do globálního minima. Tomuto míčku je potřeba dát správný směr, rychlost a momentum aby překonal všechny překážky, nikde se nezasekl a co nejkratší možnou cestou se dostal do globálního minima. (Chollet, [2018]) (Kwiatkowski, 2021)



Obrázek 9 - Gradientní sestup - Zdroj:(Rai, 2017)

Typy gradientního sestupu

Dávkový gradientní sestup zpracovává najednou celý tréninkový dataset pro aktualizaci parametrů sítě, respektive pro výpočet gradientu. Tento přístup zajišťuje stabilní konvergenci, avšak nemusí být zcela jisté, že konvergence proběhne k zcela ideálnímu řešení. Tento přístup je také velmi náročný na operační paměť, vyžaduje totiž aby byl celý dataset během trénování uložen v operační paměti, není tudíž vhodný pro velmi rozsáhlé datasety.

Stochastický gradientní sestup je extrémem v opačném směru. Parametry totiž aktualizuje po každém jednotlivém dopředném průchodu sítí. Lze tedy u tohoto přístupu říci že velikost dávky je 1. To jej činí přesnějším v dosažení ideálního minima gradientu, protože gradient přepočítává v každé iteraci. Tato skutečnost však stochastický gradientní sestup činí velmi výpočetně náročným, a navíc může gradient díky pravidelným aktualizacím obsahovat hodně šumu, který algoritmu nedovolí plynule postupovat směrem dolů k optimálnímu výsledku.

Mini-dávkový gradientní sestup je kompromisem mezi dávkovým a stochastickým gradientním sestupem. V této verzi algoritmu jsou parametry aktualizovány po malých dávkách, trénovacího datasetu. Velikost těchto dávek se pohybuje od nižších desítek až po stovky záznamů. Dávkováním datasetu dospěje tento algoritmus k efektivním méně pravidelným aktualizacím parametrů než u stochastického gradientního sestupu a dostatečně se přiblíží stabilitě dávkového gradientního sestupu bez nutnosti uložení celého datasetu v operační paměti.(Donges, 2023)

3.12 Parametry a hyperparametry

V oblasti strojového učení a hlubokého učení se často pracuje s pojmy "parametry" a "hyperparametry". Tyto pojmy hrají v této oblasti klíčovou roli konkrétně v procesu učení. Přestože se pojmy často používají společně jejich významy se podstatou liší.

3.12.1 Parametry

Parametry jsou ve své podstatě interními proměnnými modelu, které se v průběhu procesu učení modifikují směrem ke svému ideálnímu stavu a umožňují modelu se učit ze vstupních dat. V momentě, kdy model úspěšně identifikuje optimální hodnoty pro své parametry, znamená to ukončení jeho učení. V tomto okamžiku je model vybaven k tomu, aby dělal předpovědi nebo rozhodnutí na základě nových znalostí, které učením získal. Typickým příkladem parametrů v modelech strojového učení, jsou váhy a prahové hodnoty neuronů.

Během trénování jsou parametry neustále aktualizovány a optimalizovány učícím algoritmem. Optimální hodnoty parametrů potom co neaktualizuje programátor, ale místo toho se sám model tyto parametry dynamicky učí a přizpůsobuje jejich hodnoty na základě vstupních dat. Na programátorovi je však ponechána počáteční inicializace parametrů, což je také důležitý krok učící strategie. (Brownlee, 2019b) (Pramoditha, 2022)

Modelu na začátku učení například nelze pouze nastavit výchozí parametry na fixní nulovou hodnotu, v takovém případě by učící algoritmus nefungoval správně. Parametrům je nutné rozdělit různé náhodné hodnoty a existuje několik způsobů, jak toto rozdělení provést. (Alto, 2019)

1. Čistě náhodná inicializace

Nejjednodušším způsobem nastavení parametrů může být náhodná inicializace jejich hodnot v rámci omezeného intervalu (například <-0.1,0.1>). V takovémto případě bude pravděpodobnost nastavení hodnoty parametru stejná pro každou hodnotu v rámci daného intervalu. (Brownlee, 2021)

2. Inicializace s normálním rozdělením

Tento způsob inicializace parametrů přiděluje hodnoty podle normálního rozdělení se střední hodnotou 0 a malým rozptylem. Díky tomu budou mnohem častěji vybírány hodnoty blíže středu intervalu než na jeho okrajích. (Brownlee, 2021)

3. Xavierova inicializace parametrů

Problémem předchozích způsobů inicializace parametrů je jejich interakce se sigmoidní nebo tanh aktivační funkcí. Tyto funkce v intervalu blízko nulové hodnotě nabývají téměř lineárního charakteru, jak může být patrné z grafů znázorňujících tyto funkce v kapitole o aktivačních funkcích. Proto při inicializaci vah s normálním rozdělením okolo nulové hodnoty dojde k tomu, že hodnoty vah projdou aktivační funkcí jako by byla lineární, což není žádoucí při učení modelů hlubokého učení, kde jsou nelinearity základem. Stejně tak jsou problémem příliš malé nebo příliš vysoké vstupní hodnoty pro něž sigmoidní a tanh aktivační funkce vrací hodnoty 0 nebo 1 a nic mezi tím.

Řešením může být Xavierova inicializace, představená Xavierem Glorotem v roce 2010, která využívá počet vstupů a výstupů každé vrstvy dané neuronové sítě k definování náhodného intervalu, ze kterého budou váhy pro danou vrstvu vybrány. Xavierova metoda inicializace může pracovat jak s rovnoměrným rozdělením vah, kdy funkce určí hranice intervalu, tak s normálním rozdělením, kdy funkce určí směrodatnou odchylku. Obě varianty Xavierovy jsou vyjádřeny následujícími předpisy (9). (Ng, 2023)

$$[-x, x] \qquad x = \sqrt{\frac{6}{vstupy + výstupy}} \qquad \sigma = \sqrt{\frac{2}{vstupy + výstupy}} \tag{9}$$

4. He inicializace

Xavierova inicializace byla efektivní při eliminaci problémů spojených se sigmoidními a tanh aktivačními funkcemi, avšak v modelech využívajících ReLU aktivační funkce valný efekt neměla. Řešení přinesl Kaiming He v roce 2015. Jeho metoda pracuje na základě normálního rozdělení hodnot okolo nulové hodnoty. Směrodatná odchylka se však na rozdíl od Xavierovy metody určuje pouze na základě počtu vstupů dané vrstvy. (Brownlee, 2021)

$$\sigma = \sqrt{\frac{2}{vstupy}} \tag{10}$$

3.12.2 **Hyperparametry**

Hyperparametry jsou stejně jako parametry proměnné uvnitř modelu strojového učení, avšak plní odlišnou funkci. Mají přímý vliv na to, jak probíhá učící proces a potažmo ovlivňují to, jakých hodnot mohou parametry nabývat a stejně tak ovlivňují i optimální hodnoty kterých mohou parametry nakonec dosáhnout.

Klíčový rozdíl mezi parametry a hyperparametry je ten, že zatímco parametry jsou dynamické a mění se v závislosti na učebním procesu, hyperparametry jsou statické a poskytují rámec, v němž učení probíhá. Hyperparametry své hodnoty nemění ale jsou před započetím učení nastaveny programátorem a zůstávají konstantní po celou dobu.

Běžné hyperparametry zahrnují rychlost učení, počet skrytých vrstev a neuronů v dané vrstvě, velikost dávky nebo například počet epoch. Hledání optimálních hodnot hyperparametrů a jejich vhodných kombinací pro ideální splnění řešené úlohy je iterativní proces, který je jedním z nejnáročnějších kroků při vývoji modelu strojového učení. (Pramoditha, 2022)

1. Počet skrytých vrstev

Určení ideálního počtu vrstev a neuronů pro hlubokou neuronovou síť je složitý úkol, na který neexistuje jednoduché řešení a bude jednou z prvních překážek s nimiž se při návrhu modelů strojového učení lze setkat. Každá řešená úloha je unikátní a vyžaduje specifický přístup. Existují však způsoby, jak se řešení dobrat.

Základní metodou pro zjištění ideálního počtu skrytých vrstev a neuronů v nich je prosté experimentování s různými architekturami. Při návrhu architektury hluboké neuronové sítě je třeba počítat s nejistotou a iterativně se propracovat k řešení, které pro řešenou úlohu funguje. Důležité je však důkladně ověřovat funkčnost každého navrženého řešení. Při řešení úloh hlubokého učení je pravděpodobné, že vybraná řešená úloha je unikátní a nebyla doposud vyřešena, tudíž neexistuje ani žádný návod, jak hlubokou neuronovou síť pro danou úlohu připravit, proto je opatrné experimentování často jedinou možností.

Pokud řešená úloha svou podstatou naopak není nijak průlomově unikátní a specifická, existuje šance, že mnoho výzkumníků se již pokoušelo řešit úlohy příbuzné. Z těchto řešení se lze inspirovat při nastavování vlastního modelu. Je však důležité mít na paměti že přenos těchto řešení na jinou úlohu může být komplikované a nemusí fungovat. Proto je důležité opět experimentovat a ověřovat každé řešení. Výhodou ale vskutku je že inspirace jiným řešením příbuzné úlohy může přinést solidní základ, na kterém se dá stavět řešení nové.

Intuice v rámci hlubokých neuronových sítí často radí, že čím je síť hlubší, tím lepších výsledků může dosáhnout. To nicméně prokázal i výzkum Goodfellowa, Bengia a Courvilla, který tvrdí, že větší počet skrytých vrstev umožňuje síti řešit složitější úlohy lépe. Toto zjištění naznačuje, že pro náročné úlohy by mohlo být dobrým výchozím bodem navržení složitější a hlubší architektury a následná iterativní minimalizace modelu do bodu, kdy vykazuje uspokojivé chování s minimálními nároky na zdroje. To je velký kámen úrazu, nároky na zdroje a výpočetní výkon jsou při učení velmi hluboké neuronové sítě úměrně vysoké. Návrh velmi hluboké neuronové sítě je tedy ideálním přístupem, avšak pouze pokud náklady nejsou překážkou.

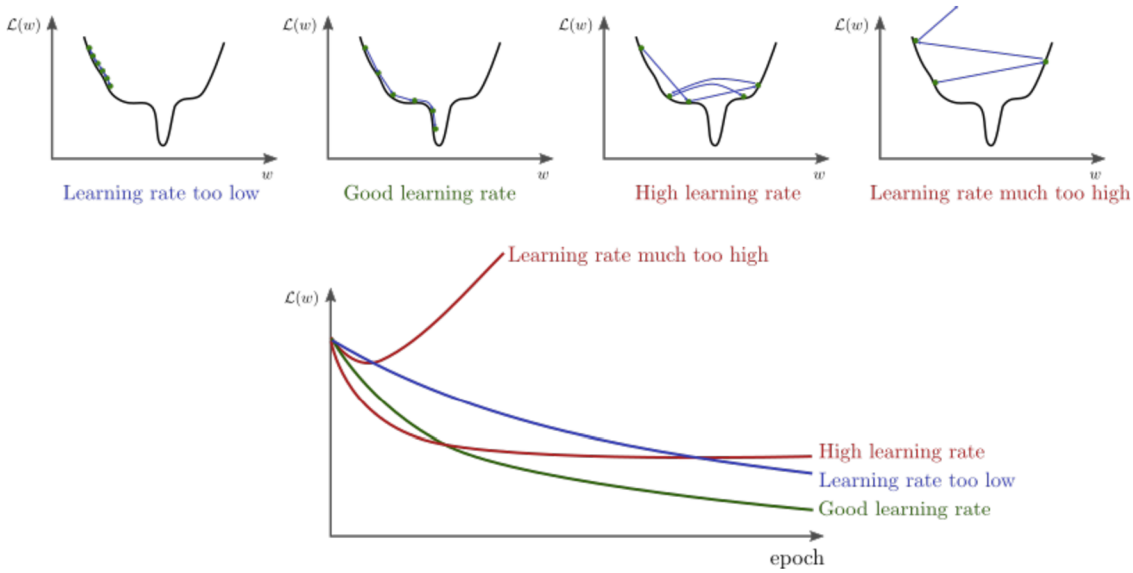
Obecně je při návrhu hloubky a složitosti sítě nutné zvolit kompromis mezi složitostí a náročností na zdroje. Pro základní koncept sítě je vhodné inspirovat se jinými řešeními či odbornou literaturou a stavět na práci někoho jiného. Pokud to není možné je dobré mít vytrénovanou intuici pro návrh hlubokých modelů a řídit se jí. Klíčové je však v každém případě každý koncept experimentálně ověřit a otestovat. Pokud po iterativním experimentování model již nijak významně nezlepšuje svůj výkon, je na místě experimentování ukončit. V takovém bodě je kontraproduktivní zvyšovat komplexitu sítě pro relativně malý nárůst její přesnosti. (Brownlee, 2019c) (Alto, 2019)

2. Rychlost učení

Rychlost učení definuje míru, s jakou jsou během učení aktualizovány váhy uvnitř sítě. Respektive hodnota tohoto hyperparametru určuje velikost kroku směřujícího k minimalizaci ztrátové funkce, což je pro efektivní učení modelu nezbytné. Správně nastavená rychlost učení zajišťuje, že model směřuje k optimálnímu nastavení vah, což samo o sobě vede k optimálnímu výkonu při řešení dané úlohy.

Volba hodnoty rychlosti učení je však často náročným krokem. Příliš vysoká rychlost učení může způsobit, že model přeskočí minimum ztrátové funkce, což vede k nestabilnímu tréninku a potenciálně brání konvergenci modelu. Na druhou stranu, příliš nízká rychlost učení může vést k velmi pomalému trénovacímu procesu. V tomto scénáři model podniká velmi malé kroky směrem k minimu, což nejen prodlužuje dobu tréninku, ale také nese riziko, že se model zasekne v lokálních minimech, čímž brání dosažení nejoptimálnějšího řešení. Cílem je nalézt ideální hodnotu tohoto hyperparametru, která umožní ztrátové funkci co nejhladší sestup do globálního minima. (Rakhecha, 2019)(Hammel, 2019)

Následující grafy znázorňují vliv různých hodnot rychlosti učení na ztrátovou funkci.



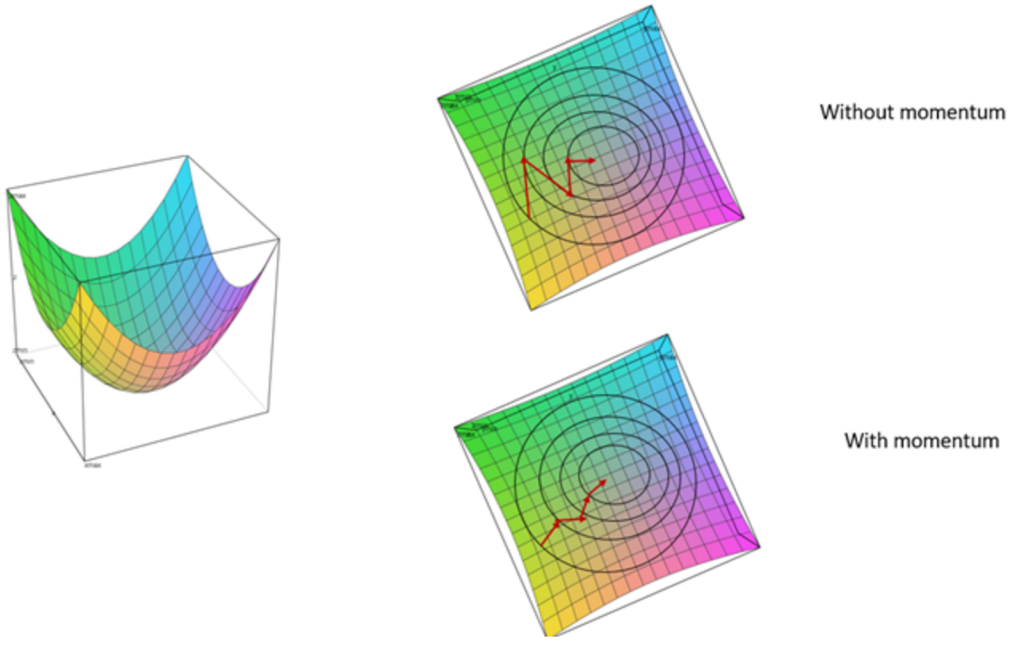
Obrázek 10 - Vliv rychlosti učení - Zdroj:(Hammel, 2019)

3. Momentum

Momentum je v rámci zpětné propagace důležitým hyperparametrem, který napomáhá efektivněji dosáhnout konvergence směrem k minimu ztrátové funkce. Na rozdíl od rychlosti učení, která určuje míru aktualizace jednotlivých vah, si momentum technicky vzato pamatuje směr, kterým byly váhy v předchozích krocích aktualizovány. Momentum následně využívá této uložené znalosti a integruje ji do dalšího kroku učení. Tento přístup v podstatě umožňuje proces učení si pamatovat směr pohybu na gradientu a pokračovat v pohybu zhruba tím směrem, kterým již dříve postupoval. Dává tak učebnímu procesu vlastnost setrvačnosti.

Momentum umožňuje rychlejší sestup k minimu, avšak výběr hodnoty tohoto hyperparametru musí být kompromisem mezi rychlostí a přesností. Vysoké momentum urychluje učební proces, ale podobně jako u vysoké hodnoty rychlosti učení existuje riziko přeskočení cílového minima kvůli nadměrné závislosti na směru předchozího sestupu. Při určení optimální hodnoty pro momentum je ideální začít s nižšími hodnotami a postupně je zvyšovat. Tato strategie vyhledává kompromis mezi rychlostí a přesností.

Využití tohoto hyperparametru se dá snadno ilustrovat na následujícím zobrazení gradientu ve tvaru paraboloidu. (Alto, 2019)



Obrázek 11 - Momentum - Zdroj:(Alto, 2019)

4. Aktivační funkce

Aktivační funkce přijímají vážený součet vstupů neuronu uvnitř sítě a mění jeho hodnotu na výstup. Vytvářejí tak v síti nezbytnou nelinearitu a umožňující zachytit ve složitých vstupních datech vzorce. Mezi typicky používané aktivační funkce patří funkce sigmoidní nebo hyperbolický tangens, dále pak funkce Softmax, nebo ReLU.

Aktivační funkce samy o sobě jsou hyperparametry, neboť je důležité rozhodnout o tom jaké funkce budou použity v různých vrstvách modelu. V praxi se pro skryté vrstvy často používá funkce ReLU kvůli rychlosti výpočtu a optimálnímu výstupu pro gradientní sestup. Naopak sigmoidní funkce nebo Softmax se často používají ve výstupní vrstvě. Softmax se specificky hodí pro úlohy klasifikace do vícero kategorií, zatímco sigmoidní aktivační funkce spíše pro binární rozdělení.

Výběr a umístění specifických typů aktivačních funkcí je pro návrh modelu hlubokého učení klíčové a u každé úlohy jde o vyvážení mezi efektivitou a požadavky konkrétních aplikací. (Radhakrishnan, 2017) (Alto, 2019)

5. Velikost dávky

Velikost dávky je hyperparametr určující počet příkladů, které jsou neuronové síti během učení představeny, než jsou váhy sítě aktualizovány. Zjednodušeně lze velikost dávky samotnou chápat jako šíři pohledu na data, v širším pohledu lze přehlédnout jemné detaily, a naopak v příliš úzkém pohledu se z dat vytrácí širší kontext. Jde o klíčový hyperparametr, na kterém závisí výpočetní čas a efektivita trénovacího procesu, a tedy i technologické nároky. Zároveň ovlivňuje výkonnost modelu.

Vztah mezi velikostí dávky a ztrátou výkonu modelů představuje komplexní problém, který nelze řešit izolovaně od dalších hyperparametrů, specificky hyperparametru rychlost učení. S konstantním nastavením rychlosti učení a variabilním nastavením velikosti dávky, lze pozorovat, že pro modely trénované s většími dávkami, nedosahuje výkon takových hodnot jako u modelů trénovaných s menšími dávkami. Konkrétně lze pozorovat snížení přesnosti a větší ztrátovost. Rychlost učení je proto klíčovým hyperparametrem, nad jehož hodnotou je nutné uvažovat vždy současně v kombinaci s velikostí dávky. Když je rychlost učení s velikostí dávky vhodně škálována, rozdíl ve ztrátě výkonu se významně snižuje. Tato závislost naznačuje, že ačkoliv velikost dávky přímo ovlivňuje výkon modelu, její dopad lze zmírnit právě kalibrací rychlosti učení a naopak.

Důležitým faktorem ovlivněným velikostí dávky je generalizace neboli schopnost modelu porozumět novým unikátním vstupům, se kterými se nesetkal při tréninku. Konkrétně tento problém vyvolává tendence uvíznutí v lokálních minimech. Modely trénované s velkými dávkami skutečně tíhnou k většímu riziku zachycení v lokálním minimu, avšak velikost dávky samotná opět není až takový problém, jak by se zdálo. Problém leží spíše v tom že s velkou velikostí dávky po několik epoch učení, nedochází k tolika aktualizacím vah modelu jako u přístupu s menšími dávkami. Řešením nedostatečné generalizace modelů proto může být navýšení počtu epoch. (Shen, 2018)

6. Počet epoch

Jedna epocha představuje jeden průchod modelu celou datovou sadou během trénovacího procesu. Počet epoch tedy zásadně ovlivňuje dobu tréninku. Počet epoch není to samé, co počet iterací, tento pojem odkazuje na počet dávek, které je třeba zpracovat k dokončení jedné epochy. Nastavení počtu epoch je silně ovlivněno povahou vstupních dat a povahou řešené úlohy. Běžnou strategií je nastavit z počátku nižší počet epoch a poté ho postupně zvyšovat. Tímto způsobem lze postupnými kroky dosáhnout optimální hodnoty, která nepovede model k přeučení.

Přeučení nastává, když se model příliš dobře naučí z tréninkových dat, včetně jejich šumu a specifických hodnot, což v důsledku omezí jeho schopnost generalizace. Model v takovém případě vykazuje vysokou přesnost na tréninkovém sub setu, ale zaostává při validaci. Jakékoliv další učení v dalších epochách tento problém pouze zhoršuje a řešením je tak pouze včasné ukončení trénovacího procesu, než k přeučení dojde. Pro takové ukončení je potřeba nastavit vhodné podmínky, které ho spustí. Vhodné je například sledovat metriky přesnosti a ztrátovosti validace a definovat si počet epoch, po které lze čekat na zlepšení hodnot těchto metrik. Zásadní je také provádět pravidelné zálohy stavů modelu, kdy metrik dosáhly nejlepších hodnot, aby následné další kroky ideální stav modelu nepřemazaly.

Na druhé straně může dojít k problému nedostatečného učení. To nastává, u příliš jednoduchých modelů, které nemají dostatečnou velikost, aby ve vybraném datasetu odhalily skryté vzorce. Model v takovém případě vykazuje nedostatečnou přesnost jak nad trénovacím sub setem, tak nad validačním. V případě problému nedostatečného učení však v určitých případech lze považovat zvýšení počtu epoch za vhodné řešení. Více epoch může

poskytnout modelu více iterací k učení a lepšímu pochopení skrytých vzorců v datech, avšak může nastat situace, kdy velikost modelu nedovoluje dosáhnout lepších výsledků ani po zvýšení počtu epoch. V takovém případě učení v dalších epochách nemá smysl, přesnost modelu neporoste a řešením je pouze modifikace architektury modelu, například přidání skrytých vrstev. (Rsvmukesh, 2023) (Brownlee, 2022) (Alto, 2019)

7. Dropout

Dalším hyperparametrem pro eliminaci přeučení může být dropout. Koncept dropoutu funguje tak, že během trénování náhodně dočasně vynechává určité výstupy vrstev. Takováto vrstva tím v zásadě pro následující vrstvu změní svůj charakter výstupu a spojení. Tento přístup přináší prospěšnou náhodnost do procesu trénování, a také nutí jednotlivé neurony, aby samostatně zpracovávaly vstupní data, čímž snižuje pravděpodobnost toho, že budou neurony příliš závislé na svých sousedech. Rozvázáním této závislosti lze vytrénovat komplexnější model a předejít tak zhoršení schopnosti modelu generalizovat.

Dropout lze použít ve všech typech skrytých vrstev a vstupní vrstvě, jeho použití však nedává smysl ve vrstvě výstupní. Samotná hodnota tohoto hyperparametru definuje pravděpodobnost, udržení výstupu neuronů v dané vrstvě během trénování. Pro vstupní vrstvu je ideální určit hodnotu dropoutu blíže hodnotě 1, tedy 100% udržení výstupu. U skrytých vrstev lze pak implementovat větší náhodnost přihlížením pravděpodobnosti blíže hodnotě 0,5.

Dropout se používá pouze v trénovací fázi. Při predikci nebo validaci jsou váhy sítě poníženy, aby byla absence dropoutu kompenzována. Tato kompenzace zajišťuje, že je výstup během testování konzistentní s výstupem sítě během tréninku. (Brownlee, 2019d) (Alto, 2019)

3.13 Tensor Flow

TensorFlow je framework pro strojové učení vyvinutý týmem Google Brain, zabývajícím se výzkumem v oblasti umělé inteligence. Nabízí komplexní sadu nástrojů pro tvorbu modelů hlubokého učení, jejich učení a nasazení v reálném provozu. Jedná se o velmi škálovatelný a flexibilní nástroj, který nabízí možnost vytvářet modely rozličné velikosti od malých modelů pro koncová zařízení po rozsáhlé modely, trénované na enormních

datasetech. Název TensorFlow vychází ze základního stavebního kamene tohoto frameworku, z tensoru. (Google LLC, 2015)

Tensor je pole ukládající číselná data v libovolném počtu dimenzí. Nejjednodušším tensorem je 0D tensor neboli skalár – jedna číselná hodnota. Přidáním jedné dimenze získáme vektor neboli 1D tensor s jednou osou. Následuje matice neboli 2D tensor s dvěma osami. Přidáním třetí osy získáme 3D tensor, který si lze představit jako jednotlivé číselné hodnoty sestavené vedle sebe a nad sebou do tvaru krychle. I 3D tensory lze pak zabalit do nadřazené dimenze a vytvořit tak 4D tensor. V hlubokém učení se tensory typicky pohybují od 0D po 5D, přičemž 5 dimenzí lze využít například při zpracování dat ve formě videa.

Veškeré fungování frameworku TensorFlow při práci s modely hlubokého učení lze tedy zredukovat na základní operace s tensory, stejně jako lze fungování počítače zredukovat na binární operace. Mezi tyto operace patří například sčítání a odčítání tensorů, násobení a dělení tensorů, transpozice, inverze a další.

Pro popsání tensoru lze použít 3 základní parametry: počet os neboli rank, tvar neboli rozměr každé osy a typ dat, která tensor obsahuje. V TensorFlow jsou nejdůležitějšími parametry tensoru shape a dtype. Parametr shape vyjadřuje tvar tensoru neboli vektor nesoucí hodnoty reprezentující velikosti každé osy vektoru. Parametr dtype vyjadřuje typ dat nesených tensorem. (Chollet, [2018])

TensorFlow pro učení modelů zpracovává algoritmus gradientního sestupu a zpětné propagace. Při řešení těchto algoritmů využívá takzvané automatické diferenciace, díky níž dokáže gradientní sestup a zpětnou propagaci provádět pro libovolný počet trénovatelných parametrů. Při automatické diferenciaci jsou při dopředném průchodu sítí zaznamenány operace v pořadí, jak jdou za sebou na gradientní pásku. Při zpětné propagaci je pak průchod páskou obrácen a pro každou operaci nad trénovatelným parametrem je vypočten gradient.

TensorFlow používá vysokoúrovňová API tf.data a tf.keras pro zjednodušení práce s frameworkem. Rozhraní tf.data zahrnuje funkce pro manipulaci s daty a hodí se při přípravě datasetu a návrhu datového kanálu dat pro model. Naopak rozhraní tf.keras poskytuje komplexní sadu funkcí pro tvorbu samotných modelů strojového učení. Toto rozhraní je v zásadě nativní implementací samostatné knihovny Keras, která bude podrobněji popsána v další kapitole.

Pro optimalizaci trénovacího procesu a následný export modelů převádí TensorFlow python kód na speciální TensorFlow optimalizovaný graf. TensorFlow je sice v základu použitelný jako python knihovna, avšak pomocí rozhraní tf.function lze dosáhnout převodu python kódu na optimalizovaný graf, reprezentující kompletní převedený kód. Pro určení kódu, který má být takto převeden stačí pouze tento kód uzavřít do anotace rozhraní tf.function. Při prvním spuštění takového kódu je vytvořen zmíněný graf, který se následně v každém dalším průchodu spouští namísto čistého python kódu, dokud není změnou kódu vynucena i změna grafu. Díky tomu lze u velkorozměrných modelů dosáhnout navýšení výkonu. Dalším benefitem grafů je také jejich přenositelnost. Grafy totiž lze exportovat a následně nasadit na různých platformách jak na serverech, tak i na koncových zařízeních. (Google LLC, 2015)

3.14 **Keras**

Keras je softwarová knihovna, vyvinutá výzkumníkem Francois Cholletem v roce 2015, sloužící jako rozhraní pro práci s neuronovými sítěmi nad frameworkem TensorFlow ale nově i nad frameworky JAX nebo PyTorch. Byla navržena tak, aby vývojářům a výzkumníkům dovolovala rychle experimentovat s hlubokými neuronovými sítěmi a rychle iterovat různá řešení vybraných úloh hlubokého učení. Myšlenka Kerasu spočívá v maximálním usnadnění práce s neuronovými sítěmi bez kompromisů v rámci flexibility nebo výkonu což nejen demokratizuje hluboké učení a činí jej dostupnějším pro širší spektrum vývojářů, ale také výrazně urychluje přechod od myšlenky k výsledku a podporuje iterativní proces vývoje, který je doméně hlubokého učení vlastní.

Keras nabízí širokou škálu předdefinovaných vysokoúrovňových modulů, vytvořených abstrakcí složité matematické logiky a hardwarové optimalizace, které lze volně spojovat mezi sebou a vytvářet tak velmi snadno modely hlubokého učení.

Základními komponentami Kerasu jsou vrstvy a modely. Základním modelem je sekvenční, který představuje sestavu vrstev, které navazují jedna na druhou neboli typický dopředný model hlubokého učení s jednou vstupní vrstvou, jednou výstupní vrstvou a libovolným počtem skrytých vrstev.

Pro vytváření komplexnějších topologií, například nelineárních nebo s vícero vstupy, lze použít *Keras functional API,* které nabízí možnost vytváření modelů s libovolně komplexní architekturou nezávislou na sekvenčním modelu. Keras také nabízí možnost vytvářet zcela nové vlastní vrstvy a modely děděním rodičovských tříd, a tím tak nabízí výzkumníkům maximální flexibilitu a možnost vytvářet skutečně komplexní modely na míru řešené úloze.

Keras za programátora obstarává veškerou komunikaci s frameworkem zvoleným pro zpracování požadavků, a i běh programu na přiděleném hardwaru. Zároveň na úrovni Kerasu nezáleží na tom, zda model běží na CPU, GPU nebo TPU. Na programátorovi je pouze volba prostředí, kde model poběží, přičemž prostředí lze snadno změnit i během vývoje. (Chollet, 2015)

4 Vlastní práce

V této části práce bude pojednáno o praktickém ověření poznatků nasbíraných v předchozí teoretické části. Poznatky budou ověřeny na úloze klasifikace zvukových záznamů pomocí hlubokého učení. Při zpracování úlohy bude splněn požadavek zadání této diplomové práce, a to sice použití dat volně dostupných na internetu pro trénování výsledného modelu hlubokého učení.

V následujících několika kapitolách bude podrobně představena úloha pro jejíž účel bude model hlubokého učení sestaven. Bude popsáno, jakým způsobem byl vybrán zvolený dataset pro trénování modelu a jeho jednotlivé části. Bude popsáno, jakým způsobem byl z datasetu vybrán subset dat pro účely zvolené úlohy. Dále bude rozebráno, jakým způsobem byla z datasetu nesoucího metadata vytěžena data pro stažení konkrétních audio souborů z internetu a jakým způsobem byla tato data transformována pro účely řešené úlohy. Následně bude popsán algoritmus zpracovávající stažené audio soubory do formátu zpracovatelného modelem hlubokého učení. Nakonec bude popsán návrh samotného hlubokého modelu a proces učení tohoto modelu na připraveném datasetu.

4.1 Popis datasetu

Pro účely úlohy řešené v rámci této práce byl zvolen dataset ručně anotovaných zvukových záznamů AudioSet. Na tomto konkrétním datasetu spolupracovaly týmy zabývající se strojovým porozuměním video a audio obsahu pod firmou Google.

Základním principem tohoto datasetu je jeho zdroj dat, tedy veřejná internetová platforma pro šíření uživatelského video obsahu YouTube taktéž pod záštitou společnosti Google. Jelikož YouTube nabízí streamování videí zdarma všem uživatelům, je tím zajištěna kompletní otevřenost AudioSetu všem, kteří jej chtějí využít. Dataset tak splňuje požadavek zadání této práce, aby data využitá pro úlohu hlubokého učení byla dostupná zdarma na internetu.

AudioSet se skládá ze dvou částí:

1. Slabě anotované záznamy

   Původní částí AudioSetu je dataset sestávající ze slabě anotovaných záznamů. Každý záznam je dlouhý 10 sekund a obsahuje vždy jeden a více štítků, přičemž je v rámci této části datasetu pravidlem, že záznamy typicky obsahují více štítků

najednou. To je dáno délkou záznamů a slabým způsobem anotování. Ve slabě anotovaném datasetu lidští anotátoři ke každému desetisekundovému záznamu přiřazují štítky, které se v rámci záznamu objevují. Dalo by se tedy říci, že základní rozlišovací jednotkou tohoto datasetu je dané desetisekundové rozmezí a blíže na první pohled nelze odhalit, kde se v daném záznamu přesně dané štítky vyskytují a který ze štítků je v záznamu nejprominentnější.

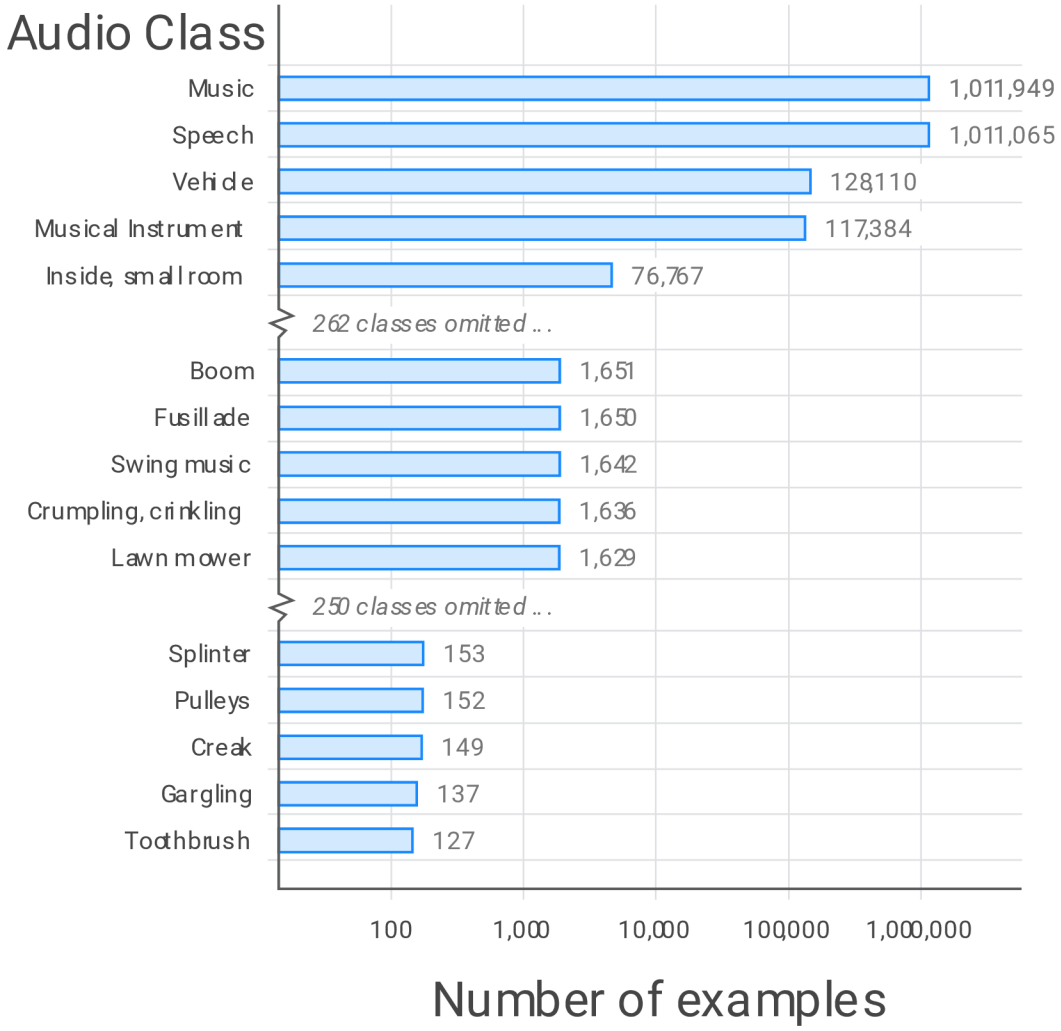
2. Silně anotované záznamy

   Novým dodatkem k původnímu datasetu slabě anotovaných záznamů je nový dataset obsahující záznamy, u nichž lidští anotátoři přesněji vymezili jednotlivé zvukové kategorie. V tomto silně anotovaném datasetu figurují záznamy nesoucí vždy jeden štítek. Platí však, že různé záznamy mohou být součástí jednoho zdrojového videa. Tímto datasetem tak autoři AudioSetu silněji vymezují temporální zasazení každé zvukové kategorie. Každý záznam v silně anotovaném datasetu je dlouhý jednu sekundu.

AudioSet sám o sobě nenese samotná data na nichž lze trénovat model hlubokého učení. Součástí datasetu jsou pouze metadata, která nesou základní informace o jednotlivých zvukových záznamech a jednotlivých kategoriích zvuků, které se v záznamech objevují. Získání samotných dat bude předmětem jedné z následujících kapitol (viz kapitola Stažení dat).

AudioSet v původní podobě obsahuje 527 kategorií zvuků, které se v záznamech objevují. Tyto kategorie se vyskytují napříč více než dvěma miliony videí, přičemž prominence kategorií je velice variabilní. Některé kategorie jsou přítomny maximálně v desítkách, čí stovkách záznamů a jiné pokrývají téměř polovinu všech anotovaných záznamů.

Zastoupení kategorií lze přehledně ilustrovat na následujícím histogramu z webové stránky AudioSetu (Obrázek 12).



Obrázek 12 - Přehled zastoupení audio kategorií v datasetu - Zdroj: (Google LLC, 2017)

Základní slabě anotovaný dataset obsahuje tři základní csv soubory.

- Vyvážený trénovací subset (balanced_train_segments.csv) obsahující 22 160 záznamů v němž je každá zvuková kategorie zastoupena minimálně 59 záznamy. Navíc se tento subset drží pravidla poskytnout minimální počet záznamů s minimálním počtem celkových záznamů.
- Evaluační subset (eval_segments.csv) nesoucí dodatečných 20 383 záznamů. Tyto záznamy vykazují vyšší míru sdílení kategorií než vyvážený subset, proto je záznamů v tomto evaluačním subsetu méně ačkoli pravidlo zastoupení každé kategorie alespoň 59 záznamy stále platí.
- Zbytek datasetu (unbalanced_train_segments.csv) nesoucí zbylou většinu datasetu tedy 2 042 985 záznamů.

Všechny výše zmíněné soubory sdílí jeden stejný formát. Zde je uveden příklad formátu na jednom ze souborů, konkrétně balanced_train_segments.csv

```
# Segments csv created Sun Mar  5 10:54:31 2017
# num_ytids=22160, num_segs=22160, num_unique_labels=527, num_positive_labels=52882
# YTID, start_seconds, end_seconds, positive_labels
--PJHxphWEs, 30.000, 40.000, "/m/09x0r,/t/dd00088"
--ZhevVpy1s, 50.000, 60.000, "/m/012xff"
--aE2O5G5WE, 0.000, 10.000, "/m/03fwl,/m/04rlf,/m/09x0r"
--aO5cdqSAg, 30.000, 40.000, "/t/dd00003,/t/dd00005"
--aaILOrkII, 200.000, 210.000, "/m/032s66,/m/073cg4"
--cB2ZVjpnA, 30.000, 40.000, "/m/01y3hg"
--ekDLDTUXA, 30.000, 40.000, "/m/015lz1,/m/07pws3f"
-0DLPzsiXXE, 30.000, 40.000, "/m/04rlf,/m/07qwdck"
-0DdlOuIFUI, 50.000, 60.000, "/m/0130jx,/m/02jz0l,/m/0838f"
-0FHUc78Gqo, 30.000, 40.000, "/m/02w4v,/m/04rlf"
-0O3e95y4gE, 100.000, 110.000, "/m/07r4wb8,/t/dd00125"
-0SdAVK79lg, 30.000, 40.000, "/m/0155w,/m/01lyv,/m/0342h,/m/042v_gx,/m/04rlf,/m/0gg8l"
-0Vl4HyWRk8, 410.000, 420.000, "/m/085jw,/m/0l14l2"
-0mG4W5Hlq8, 270.000, 280.000, "/m/04rlf,/m/05fw6t,/m/07r4k75,/m/09x0r,/m/0ytgt"
-0mjrMposBM, 80.000, 90.000, "/m/04zmvq"
-11LhdJgBb8, 30.000, 40.000, "/m/04rlf,/m/07qn4z3"
-1LrH01Ei1w, 30.000, 40.000, "/m/02p0sh1,/m/04rlf"
-1TLtjPtnms, 10.000, 20.000, "/m/03lty,/m/04rlf,/m/07szfh9"
```

Příklad kódu 1 - Formát vyváženého slabě anotovaného datasetu - Zdroj: Autor

Z formátu jsou patrné základní parametry záznamů, s kterými lze pracovat.

- YTID

  Tento parametr je unikátním identifikátorem videa uloženého na platformě YouTube. S pomocí tohoto identifikátoru lze zkonstruovat URL adresu odkazující na dané video. Z tohoto odkazu lze následně také video stáhnout, jak bude popsáno v kapitole o stažení dat.
- start_seconds

  Začátek vybraného audio segmentu z daného videa v sekundách.
- end_seconds

  Konec vybraného audio segmentu z daného videa v sekundách.
- positive_labels

  Tento parametr nese informaci o tom, jaké kategorie zvuků – štítky – se vyskytují v daném záznamu. Parametr má formát řetězce ohraničeného uvozovkami. Každý jednotlivý štítek je potom oddělen čárkou.

Na tomto příkladu si lze povšimnout několika aspektů, které definují celý slabě anotovaný subset. Ačkoli je toto jen malý výřez, vztahují se tyto aspekty na celý slabě anotovaný subset.

1. YTID se neopakují. To znamená že každý segment se nachází v unikátním YouTube videu.
2. Hranice segmentů vždy vedou k ohraničení přesně deseti sekund videa.
3. Každý segment obsahuje minimálně jeden štítek, avšak zpravidla většinou obsahuje dva a více štítků.

Posledním důležitým souborem vztahujícím se k slabě anotovanému datasetu je class_labels_indices.csv. Tento soubor je v zásadě slovníkem pro slabě anotovaný dataset. Obsahuje tři parametry oddělené čárkami.

- index

  Inkrementální číselný identifikátor každé kategorie. Pořadí kategorií není nijak specificky definováno.
- mid

  Řetězec reprezentující identifikátor kategorie – štítek. Tento štítek se vyskytuje ve všech souborech jako zástupný identifikátor dané kategorie.
- display_name

  Parametr reprezentující lidsky čitelné pojmenování kategorie

Příklad formátu souboru:

```
index,mid,display_name
0,/m/09x0r,"Speech"
1,/m/05zppz,"Male speech, man speaking"
2,/m/02zsn,"Female speech, woman speaking"
3,/m/0ytgt,"Child speech, kid speaking"
4,/m/01h8n0,"Conversation"
5,/m/02qldy,"Narration, monologue"
6,/m/0261r1,"Babbling"
7,/m/0brhx,"Speech synthesizer"
8,/m/07p6fty,"Shout"
9,/m/07q4ntr,"Bellow"
10,/m/07rwj3x,"Whoop"
11,/m/07sr1lc,"Yell"
12,/m/04gy_2,"Battle cry"
13,/t/dd00135,"Children shouting"
14,/m/03qc9zr,"Screaming"
15,/m/02rtxlg,"Whispering"
16,/m/01j3sz,"Laughter"
```

Příklad kódu 2 - Formát slovníku slabě anotovaného subsetu - Zdroj: Autor

Na rozdíl od slabě anotovaného subsetu, silně anotovaný subset obsahuje pouze dva základní soubory popisující data a jeden soubor popisující štítky. Soubory popisujícími data jsou tyto.

- Trénovací subset (audioset_train_strong.tsv) obsahující 934 821 záznamů napříč 103 463 audio soubory. Napříč tímto souborem se nachází 447 unikátních štítků z nichž některé nefigurují v původním slabě anotovaném subsetu. Konkrétně počet štítků, které tento soubor sdílí s původním slabě anotovaným subsetem je 376.
- Evaluační subset (audioset_eval_strong.tsv) obsahující 139 538 záznamů napříč 16 996 audio soubory. Obsahuje 416 unikátních štítků z nichž 381 je sdíleno s původním slabě anotovaným datasetem. Navíc tento evaluační subset obsahuje 9 štítků, které nejsou přítomny v silně anotovaném trénovacím souboru.

Na rozdíl od slabě anotovaného subsetu, silně anotovaný subset má mírně odlišný formát. Prvním rozdílem je jiný typ souboru (tsv namísto csv), v němž jsou parametry odděleny tabulátorem nikoliv čárkou. Tento formát lze používat stejně tak snadno jako csv, nebo jej v zájmu konzistentnosti snadno transformovat do formátu csv jak bude popsáno v kapitole příprava datasetu.

Rozdíl subsetů však nalezneme i v samotných parametrech souborů. Sobory silně anotovaného subsetu nesou následující parametry:

- YTID

  Parametr nesoucí unikátní identifikátor YouTube videa, z něhož pochází referovaný audio segment má v tomto případě jiný formát než u slabě anotovaného subsetu. V původním subsetu parametr obsahuje pouze unikátní identifikátor, zde však je k identifikátoru podtržítkem připojena informace o tom, v jakém bodě videa začíná větší segment, v němž jsou parametry start_time_seconds and end_time_seconds vyhraničeny menší specifické subsegmenty v podrobnějším přibližně jednosekundovém rozlišení. Prakticky tak tento dodatek k YTID slouží jako hodnota posunu ohraničujících parametrů od začátku videa.
- start_time_seconds

  Parametr reprezentující začátek subsegmentu bez posunu.
- end_time_seconds

  Parametr reprezentující konec subsegmentu bez posunu.
- label

  Parametr nesoucí informaci o štítku přítomném v daném subsegmentu. Základní vlastností silně anotovaného datasetu je to, že každému segmentu

reprezentovanému jedním řádkem jednoho z výše zmíněných souborů odpovídá vždy jeden štítek.

Formát silně anotovaného subsetu tedy vypadá následovně:

```
YTID    start_time_seconds  end_time_seconds    label
NQNTnl0zaqU_70000   7.999   8.933   /m/0c1dj
NQNTnl0zaqU_70000   9.063   10.000  /m/0c1dj
4PPmyY_-YrA_30000   7.983   8.161   /m/07qjznt
O35jXasNYxc_30000   7.076   7.322   /m/01b_21
O35jXasNYxc_30000   7.307   7.921   /m/0dgw9r
O35jXasNYxc_30000   8.205   9.596   /m/01b_21
O35jXasNYxc_30000   9.663   10.000  /m/0dgw9r
cUAe_N9oODs_0   0.148   2.329   /m/07qf0zm
cUAe_N9oODs_0   0.363   0.883   /m/02zsn
```

Příklad kódu 3 - Formát silně anotovaného datasetu - Zdroj: Autor

Stejně tak jako slabě anotovaný subset i silně anotovaný subset má vlastní slovník štítků. V tomto subsetu opět ve formátu tsv a jak již bylo zmíněno obsahuje tento slovník i štítky, které se v původním subsetu nevyskytují, a naopak neobsahuje některé štítky z původního subsetu. Slovník štítku mid_to_display_name.tsv má následující podobu:

```
/g/11b630rrvh   Kettle whistle
/g/122z_qxw Firecracker
/m/01280g   Wild animals
/m/0150b9   Change ringing (campanology)
/m/015jpf   Dial tone
/m/015lz1   Singing
/m/015p6    Bird
/m/0160x5   Digestive
/m/01g90h   Stomach rumble
/m/01h3n    Bee, wasp, etc.
/m/01hnzm   Ringtone
/m/01hsr_   Sneeze
```

Příklad kódu 4 - Formát slovníku silně anotovaného datasetu - Zdroj: Autor

4.2 Představení řešené úlohy

Pro aplikaci znalostí představených v teoretické části bude v rámci práce řešena úloha klasifikace audio záznamů pomocí hluboké neuronové sítě. Klasifikace audio záznamů spočívá v rozřazení jednotlivých audio záznamů do několika předem definovaných kategorií podle obsahu daného audio záznamu.

Aby bylo toto rozřazení možné je nutné hlubokou neuronovou síť natrénovat na datovém souboru, který ke každému jednotlivému audio záznamu přiřazuje kategorie zvuků, které se v daném audio záznamu vyskytují. Tyto kategorie jsou typicky reprezentovány štítkem v podobě unikátního identifikátoru dané kategorie. Štítky v podobě unikátních identifikátorů jsou typicky popsány ve zvláštním souboru, který jim přiřazuje lidsky čitelnou

formu. Jak již bylo v předchozí kapitole zmíněno, pro tuto úlohu byl vybrán dataset lidmi anotovaných audio záznamů AudioSet.

Specifikem řešené úlohy je to, že z velkorozměrného datasetu bude vybráno pouze několik štítků, které budou klasifikovány. Cílem úlohy bude ověřit, zda je možné provést takový výběr z velkorozměrného datasetu a natrénovat na tomto výběru model hlubokého učení s uspokojivými výsledky či nikoli. Za uspokojivý výsledek bude považován stav, kdy model v procesu učení vykáže pozitivní trend metrik měřících úspěšnost učení a ideálně následně na evaluačním datasetu prokáže uspokojivou míru predikce.

Konkrétní klasifikační úlohou bude rozpoznání nebezpečných či problematických zvuků pro účely monitorovacího bezpečnostního systému. Tento problém bude zahrnovat rozpoznání několika tříd zvuků jevů ohrožujících bezpečnost osob v budovách či veřejných prostorech. Mezi tyto zvuky lze řadit například: křik, střelbu, exploze, tříštění skla, nebo auto alarm. Na výběru konkrétních zvuků bude záviset jejich zastoupení v obou subsetech.

V rámci úlohy bude nutné učinit několik zásadních kroků, které povedou ke konečnému cíli, tedy učení modelu a vyhodnocení výsledků učení.

1. Analýza a příprava datasetu

   Dataset v hrubé formě, bude potřeba transformovat do podoby, s kterou půjde pracovat. To v tomto případě zahrnuje:

   1. Transformaci všech souborů slabě i silně anotovaného subsetu do jednotného formátu. Ideálně csv s jednotným formátováním a oddělovači.
   2. Analýzu a vyfiltrování štítků společných pro slabě anotovaný subset a silně anotovaný trénovací subset.
   3. Ověření aplikovatelnosti vyfiltrovaných štítků na zvolenou úlohu.
2. Stažení dat

   Následně bude nutné zdrojová videa uložená na platformě YouTube stáhnout a extrahovat z nich segmenty uložené v souborech s metadaty.
3. Analýza a zpracování stažených dat

   Stažená data bude nutné analyzovat a transformovat do podoby zpracovatelné hlubokou neuronovou sítí. Bude nutné vytvořit mechanismus, který hrubé audio soubory dokáže načíst, transformovat do podoby spektrogramu a s připojenými štítky z nich dokáže sestavit tensorovou datovou sadu, zpracovatelnou neuronovou

sítí. Zároveň tento mechanismus musí transformovat i samotné štítky, původně uložené jako řetězce do podoby tensoru. Součástí tohoto kroku bude i analýza spektrogramů. Bude důležité ověřit že všechny spektrogramy obsahují požadovaná data v přijatelné kvalitě. Případně musí být závadná data z datasetu vyfiltrována.

4. Návrh modelu a učení

   Po načtení všech dat bude na řadě nejkomplexnější úkol. Návrh modelu a následné iterativní trénování modelu na načtených datech a validace. V tomto kroku budou nejsilněji využity poznatky z teoretické části k dosažení efektivního učení modelu iterativním laděním hyperparametrů a architektury modelu.

Veškeré programování v rámci úlohy bude provedeno v programovacím jazyce Python. Zároveň budou využity populární knihovny pandas, numpy nebo matplotlib. Datový kanál pro zpracování hrubých audio souborů a štítků do podoby tensorového datasetu bude zhotoven prostřednictvím frameworku TensorFlow. Model hluboké neuronové sítě bude vytvořen s pomocí knihovny Keras, která je integrována do frameworku jako API.

4.3 Příprava datasetu

V této části se zaměříme na přípravu datasetu do podoby, s níž půjde pohodlněji pracovat. Oba subsety používají jiný způsob uložení dat, proto prvním krokem bude převedení tsv souborů ze silně anotovaného datasetu na csv jako je tomu u slabě anotovaného datasetu. S oběma subsety nakonec budeme chtít pracovat v tandemu při načtení datasetu pro model proto je logickým krokem si práci na začátku usnadnit unifikací formátů.

Pro transformaci souborů do cílového formátu byla použita Python knihovna pandas. Kód zde nebude uveden, neboť se jedná o rutinní úkony, které lze replikovat následováním dokumentace zmíněné knihovny. Pro ilustraci bude na následujícím příkladu uveden výsledný formát souborů metadat audio záznamů. Všechny výsledné soubory jsou ve formátu csv a mají totožné parametry YTID, start_seconds, end_seconds a positive_labels.

```
YTID,start_seconds,end_seconds,positive_labels
--cB2ZVjpnA,30.000,40.000,/m/01y3hg
-0DdlOuIFUI,50.000,60.000,"/m/0130jx,/m/02jz0l,/m/0838f"
-0FHUc78Gqo,30.000,40.000,"/m/02w4v,/m/04rlf"
```

Příklad kódu 5 - Formát upraveného slabě anotovaného subset - Zdroj: Autor

```
YTID,start_seconds,end_seconds,positive_labels
LvNUyQ3xuAQ_0,0.596,5.677,/m/015jpf
b0RFKhbpFJA_30000,0.0,10.0,/m/03m9d0z
b0RFKhbpFJA_30000,8.534,9.156,/t/dd00092
NQNTnl0zaqU_70000,0.0,0.103,/m/07rdhzs
```

Příklad kódu 6 - Formát upraveného silně anotovaného subsetu - Zdroj: Autor

Ačkoli byly soubory napříč datasetem uvedeny do stejného formátu, stále se liší ve dvou základních aspektech. Prvním z nich je fakt, že silně anotovaný dataset k parametru YTID připojuje za podtržítkem ještě informaci o posunu parametrů start_seconds a end_seconds. V rámci přípravy datasetu můžeme i tento rozdíl eliminovat. Jak lze vypozorovat, všechny řádky silně anotovaných souborů obsahují posun, tedy i ty které jej nepotřebují. V takovém případě je posun vyjádřen nulou stále oddělenou podtržítkem. Jednoduše pak stačí pro všechny řádky silně anotovaných souborů rozdělit parametr YTID podle posledního podtržítka nacházejícího se v tomto parametru. První polovinou rozděleného parametru přepíšeme původní YTID aby odpovídalo formátu u slabě anotovaného subsetu a s druhou polovinou budeme dále pracovat tak, že extrahovanou hodnotu vydělíme tisícem abychom z milisekund získali sekundy a tuto hodnotu přičteme k parametrům ohraničujícím daný audio segment.

Výsledný soubor tedy vypadá stejně jako soubory slabě anotovaného datasetu. Přetrvává však ještě jeden rozdíl. Vícero segmentů silně anotovaného subsetu na rozdíl od slabě anotovaného subsetu totiž může pocházet z jednoho zdrojového videa. Tento problém bude nutné zohlednit v následující kapitole.

V rámci přípravy datasetu byla navíc provedena analýza slovníkových souborů, obsahujících informace o štítcích. Pro potřeby řešené úlohy byly porovnány slovníky slabě anotovaného i silně anotovaného subsetu a byl z nich proveden výběr štítků spadajících do obou subsetů současně. Výsledný soubor vzniklý průnikem subsetů, obsahuje parametry id, name, name_strong spojující štítek s popisem z obou datasetů. Z tohoto souboru byl proveden výběr konkrétních štítků, pro něž bude úloha uzpůsobena.

```
mid,name,name_strong
/m/028v0c,Silence,Silence
/m/03qc9zr,Screaming,Screaming
/m/032s66,"Gunshot, gunfire","Gunshot, gunfire"
/m/014zdl,Explosion,Explosion
/m/07qlf79,Spray,Spray
/m/02mfyn,Car alarm,Car alarm
/m/07rn7sz,Shatter,Glass shatter
```

Příklad kódu 7 - Původní množina zvuků pro klasifikaci - Zdroj: Autor

Tyto štítky reprezentují zvukové události, znamenající potenciální ohrožení osob nebo majetku, které mohou být subjekty sledovanými potencionálním bezpečnostním systémem sledujícím zvuky v okolí s cílem monitorovat a reportovat tyto události.

4.4 Stažení dat

Než bude možné započít jakoukoli další práci, je nutné nejprve získat data, ze kterých se bude model hlubokého učení nakonec učit. AudioSet sám o sobě data neobsahuje. Obsahuje pouze metadata, tedy informace popisující data. Data samotná jsou uložena jako videa na veřejné platformě s uživatelským video obsahem YouTube.

Metadata uložená v datasetu obsahují informace o tom, z jakého YouTube videa pochází dané segmenty v podobě parametru YTID, který je unikátním identifikátorem každého takového videa. Pomocí tohoto identifikátoru lze snadno odvodit URL adresu na níž se dané video nachází. Z této adresy pak lze video stáhnout do lokálního úložiště jako soubor mp4. Z tohoto souboru následně lze extrahovat segmenty, které se v něm nachází.

U slabě anotovaného subsetu by problém stažení dat nebyl až tak složitý, avšak komplikuje se s požadavkem na stažení obou subsetů, tedy slabě i silně anotovaného. Subsety se i po provedení úprav z předchozí kapitoly liší v jednom aspektu. A sice v tom, že v případě silně anotovaného subsetu může z jednoho zdrojového videa pocházet i několik segmentů zároveň.

V rámci této práce bude vytvořen algoritmus pro stažení obou datových sad, který se bude snažit o maximální efektivitu při stahování dat, tak aby nedocházelo ke zbytečné redundanci, tedy k opakovanému stahování stejných videí.

V první části algoritmu bude nutné připravit metadata použitelná ke stažení dat. Přípravu metadat bude mít na starosti funkce, která jako parametry přijme: objekt s cílovými štítky, cestu k csv souboru s metadaty, cestu k adresáři, kam budou data uložena, cílový typ stažených souborů a parametr pro limit stahovaných segmentů pro každý štítek. Funkce provede následující kroky:

1. Načtení souboru s metadaty
2. Vyfiltrování štítků, které nespadají do předem definované množiny
   Tento krok je důležitý, neboť finální model hlubokého učení, budeme učit rozpoznávat pouze štítky z dané množiny. Každý další štítek, by znamenal rozšíření

výstupu modelu a jelikož ano o takový výstup nestojíme, můžeme štítky mimo definovanou množinu zahodit.

3. Sestavení cesty pro uložení souboru

   Pro každý řádek metadat sestavíme z jednotlivých parametrů, cestu s názvem souboru, kde bude cílový stažený audio segment uložen. Tato cesta k souboru poslouží k jeho stažení a následnému načtení.

4. Metadata s novými parametry uložíme jako nový soubor do speciální adresáře, kde budeme uchovávat vyfiltrované datasety. Tato uložená metadata budeme v dalším kroku používat pro načtení stažených souborů.

5. Dále filtrujeme metadata, která předáme algoritmu pro stažení dat. Vybereme pro každý štítek maximálně tolik vzorků, kolik definuje parametr limit. Z plných metadat pak vyfiltrujeme ty záznamy, které jsou již případně stažené.

Takto vyfiltrovaná metadata předáme algoritmu pro stažení dat. Na dalším příkladu je ukázka výsledného csv souboru, který funkce pro přípravu metadat vyprodukovala.

```
YTID,start_seconds,end_seconds,positive_labels,file_path,data_path,file_type
3zOfKHtXxws,34.181,34.75,/m/02mfyn,data/strong/3zOfKHtXxws_34.181-34.75.wav,data/strong,wav
tAN1g7ZhE4c,0.115,5.106,/m/07qlf79,data/strong/tAN1g7ZhE4c_0.115-5.106.wav,data/strong,wav
3tDgrjEX1mE,30.379,32.962,/m/014zdl,data/strong/3tDgrjEX1mE_30.379-32.962.wav,data/strong,wav
3tDgrjEX1mE,33.385,34.644,/m/014zdl,data/strong/3tDgrjEX1mE_33.385-34.644.wav,data/strong,wav
3tDgrjEX1mE,35.944,37.122,/m/014zdl,data/strong/3tDgrjEX1mE_35.944-37.122.wav,data/strong,wav
3tDgrjEX1mE,38.438,39.006,/m/014zdl,data/strong/3tDgrjEX1mE_38.438-39.006.wav,data/strong,wav
aCNk-6UiGLI,108.454,109.319,/m/032s66,data/strong/aCNk-6UiGLI_108.454-109.319.wav,data/strong,wav
28oK-ruIvVw,77.317,78.21,/m/03qc9zr,data/strong/28oK-ruIvVw_77.317-78.21.wav,data/strong,wav
aANx5lIAkeI,158.056,159.097,/m/032s66,data/strong/aANx5lIAkeI_158.056-159.097.wav,data/strong,wav
S7X8VIkM3oU,8.441,9.213,/m/03qc9zr,data/strong/S7X8VIkM3oU_8.441-9.213.wav,data/strong,wav
GXQWgHBe6Y0,35.299,36.142,/m/032s66,data/strong/GXQWgHBe6Y0_35.299-36.142.wav,data/strong,wav
GXQWgHBe6Y0,36.827,38.047,/m/032s66,data/strong/GXQWgHBe6Y0_36.827-38.047.wav,data/strong,wav
```

Příklad kódu 8 - Podklad pro stažení dat - Zdroj: Autor

S těmito metadaty v dalším kroku pracuje funkce pro stažení dat. Funkce seskupí všechny parametry týkající se audio segmentů podle společného YTID. Následně nad těmito seskupenými metadaty spustí v paralelním zpracování funkci pro stažení a extrakci segmentů. Paralelní zpracování je provedeno pomocí knihovny joblib.

```
def download(metadata):

    data_path = metadata.iloc[0]['data_path']
    grouped_metadata = metadata.groupby('YTID')['start_seconds', 'end_seconds',
'file_path'].apply(lambda x: list(x.itertuples(index=False, name=None))).to_dict()
    grouped_args = [(ytid, segments, data_path) for ytid, segments in grouped_metadata.items()]

    joblib.Parallel(n_jobs=-1, verbose=10)(
        joblib.delayed(process_video_segments)(args) for args in grouped_args
    )
```

Příklad kódu 9 - Funkce pro stažení dat - Zdroj: Autor

Paralelně spuštěná funkce process_video_segments následně pro každou skupinu segmentů s daným YTID stáhne video, extrahuje z něho segmenty a po extrakci všech segmentů video odstraní.

```
def process_video_segments(ytid, segments, data_path):

    downloaded = download_video(ytid, data_path)

    for start_seconds, end_seconds, file_path in segments:
        extract_audio(ytid, start_seconds, end_seconds, file_path, data_path)

    if downloaded:
        os.remove(os.path.join(data_path, f"{ytid}.mp4"))
```

Příklad kódu 10 - Funkce pro extrakci audio segmentů z videa - Zdroj: Autor

Stažení videa je provedeno s pomocí další knihovny, a to sice PyTube následujícím způsobem.

```
yt = YouTube(f'https://www.youtube.com/watch?v={ytid}')
stream = yt.streams.filter(progressive=True, file_extension='mp4').order_by('resolution').desc().first()
stream.download(filename=os.path.join(data_path, f"{ytid}.mp4"))
```

Příklad kódu 11 - Stažení videa - Zdroj: Autor

Extrakce segmentů ze staženého videa je provedena pomocí programu ffmpeg pro manipulaci s multimediálním obsahem. Z videa je příkazem extrahován daný segment a dále je extrahovanému segmentu nastaven parametr vzorkovací frekvence a počtu audio kanálů. Nastavením těchto parametrů již při stažení videa je zajištěno to, že tyto úkony následně nemusí řešit algoritmus zpracovávající stažené audio soubory do podoby tensorového datasetu a tím je výrazně ušetřen výpočetní čas.

```
cmd = [
    'ffmpeg', '-i', os.path.join(data_path, f"{ytid}.mp4"),
    '-ss', str(start_seconds), '-to', str(end_seconds),
    '-ar', str(SAMPLE_RATE),
    '-ac', str(CHANNELS),
    '-vn', file_path
]
subprocess.run(cmd, check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

Příklad kódu 12 - Extrakce audio segmentu - Zdroj: Autor

V případě řešené úlohy je vzorkovací frekvence audio souborů fixně nastavena na 16kHz a soubory mají pouze jeden audio kanál.

4.5 Zpracování stažených dat

Ve stavu, kdy jsou stažena potřebná data a jejich upravená metadata jsou uložena v novém csv souboru, lze přistoupit k dalšímu kroku. V tomto kroku bude vytvořen datový

kanál, jehož prostřednictvím bude možné stažená data načíst a transformovat do podoby, s kterou dokáže model hlubokého učení pracovat.

Datový kanál bude tvořit jedna základní zastřešující funkce, jejímž vstupem bude cesta k souboru s metadaty. Funkce tento soubor načte a na základě něho bude postupně načítat z úložiště stažené audio soubory a provádět nad nimi transformace pomocí další vnořené funkce. Tato vnořená funkce zajistí načtení surového audio souboru ve formátu wav a extrakci audio stopy. Audio stopu zasadí do fixního desetisekundového okna pro případ, že stažený soubor tuto hranici překračuje nebo jeho délka nedostačuje. Následně audio stopu transformuje do podoby spektrogramu. Zastřešující funkce na výstupu vrátí dataset ve formátu tf.data zpracovatelný modelem hlubokého učení, který každému spektrogramu přiřadí vektor štítků obsažených v dané audio stopě.

Na následujícím přikladu je vyobrazena vnořená funkce pro zpracování audio stopy do podoby spektrogramu. Důležitým aspektem této funkce je výhradní použití nativních TensorFlow nástrojů pro manipulaci s audio stoupou. Tím je dosaženo maximální efektivity, díky tomu, že TensorFlow následně dokáže veškeré nativní operace na pozadí převádět do grafové podoby a tím urychlit spouštění. Tato funkce TensorFlow je ve výchozím stavu aktivovaná, avšak byla by pro danou funkci neaktivní, pokud by funkce implementovala nástroje z jiných knihoven nebo nativní nástroje jazyka Python.

```
def process_file(file_path, labels):
    #Načtení dat
    audio_binary = tf.io.read_file(file_path)
    audio, sr = tf.audio.decode_wav(audio_binary, desired_channels=1)
    audio = tf.squeeze(audio, axis=-1)
    #Kontrola délky stopy
    max_len = int(SAMPLE_RATE * (DURATION_MS / 1000))
    audio_len = tf.shape(audio)[0]
    # ošetření prázdné audio stopy
    if audio_len == 0:
        labels = tf.zeros([len(labels)], dtype=tf.float32)
        spectrogram = tf.zeros(SPECTROGRAM_SHAPE, dtype=tf.float32)
        return spectrogram, labels
    if audio_len > max_len: # oříznutí stopy na maximální délku
        audio = audio[:max_len]
    elif audio_len < max_len: # doplnění stopy nulovými hodnotami na maximální délku
        pad_begin_len = tf.random.uniform(shape=[], minval=0, maxval=max_len - audio_len,
dtype=tf.int32)
        pad_end_len = max_len - audio_len - pad_begin_len
        audio = tf.pad(audio, [[pad_begin_len, pad_end_len]], "CONSTANT")
    #Vytvoření spektrogramu
    spectrogram = tf.signal.stft(audio, frame_length=255, frame_step=128)
    spectrogram = tf.abs(spectrogram)
    spectrogram = tf.expand_dims(spectrogram, axis=-1)
    return spectrogram, labels
```

Příklad kódu 13 - Funkce pro transformaci audio souborů na spektrogramy - Zdroj: Autor

Jak je patrné z kódu, funkce pro zpracování surových audio souborů do spektrogramů obsahuje tři zásadní kroky.

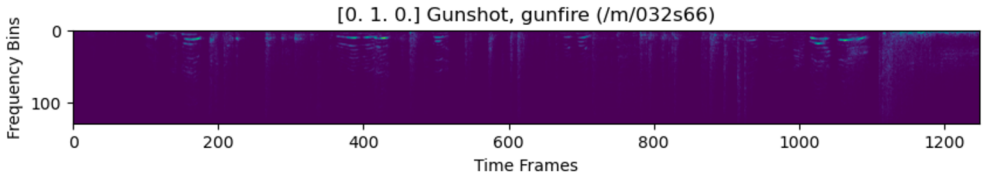
Prvním krokem je načtení audio souboru. Hrubý soubor je načten z dané cesty a jeho obsah je extrahován pomocí TensorFlow funkce pro extrakci obsahu wav souborů. Tato nativní funkce je důvodem, proč byly v předchozím kroku práce všechny soubory staženy v tomto formátu. Pro použití jiného audio formátu by bylo nutné soubory dekódovat nástroji alternativních knihoven což by výrazně zbrzdilo proces zpracování dat. Jelikož po stažení všechny audio soubory obsahují jen jeden audio kanál, je tento fakt specifikován při dekódování obsahu souboru a následně je pomocí funkce squeeze nepotřebný kanál odebrán.

Následuje krok ověření konstantní délky audio stopy. V případě, že je zvuková stopa delší než požadovaných 10 sekund, je jednoduše oříznuta. Není typické, že by stopa překračovala požadovanou délku velmi markantně, proto není problém provést jednoduchý ořez bez specifikování posunu. V opačném případě, kdy stopa nedosahuje požadované délky je však na místě se posunem zabývat. Výskyt stop kratších, než je požadováno, je větší, než výskyt stop delších ale hlavně je nutné funkci připravit i na případné zpracování silně anotovaných souborů, které požadované délky nedosahují nikdy. Posun stopy při rozšíření na požadovanou délku je náhodný, aby v datech nevznikaly nechtěné vzorce.

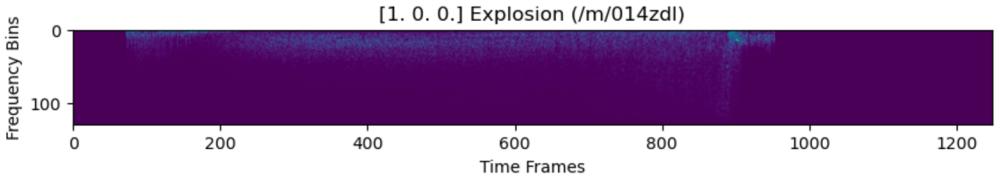
V průběhu implementace funkce pro zpracování dat bylo zjištěno že některé soubory po načtení obsahují prázdnou zvukovou stopu. Těchto souborů není mnoho, přesto mohly být příčinou dodatečné nevyváženosti datové sady, proto v případě, že má audio stopa nulovou délku, vrací funkce prázdný spektrogram i prázdné štítky, které jsou v dalším kroku vyfiltrovány. Příčinou prázdné audio stopy může být nesprávné stažení, ale tento jev se neobjevuje v datech příliš často.

V posledním kroku je audio stopa je zpracována krátkodobou Fourierovou transformací do podoby spektrogramu. Ten je následně ještě rozšířen o jednu dimenzi reprezentující jeden obrazový kanál. Tento krok je důležitý proto aby hluboká neuronová síť dokázala zpracovat jako obrazovou informaci. Úloha se tak v zásadě stává úlohou strojového vidění.

Produktem krátké Fourierovy transformace je základní spektrogram, který je zobrazen na následujících grafech 7 a 8.



Graf 7 - Základní spektrogram zvuku střelby - Zdroj: Autor



Graf 8 - Základní spektrogram zvuku exploze - Zdroj: Autor

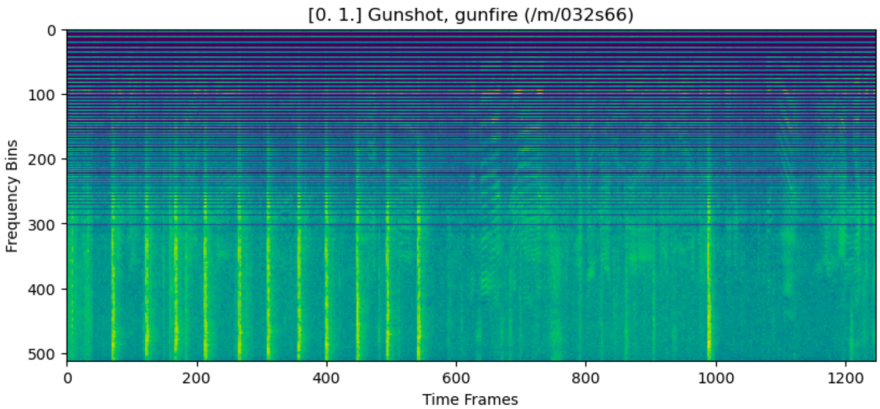
Tento spektrogram na první pohled neobsahuje příliš informací, avšak neznamená to, že, žádné nenese. Informace jsou ve spektrogramu zakódovány v podobě tensorů a neuronová síť je dokáže přečíst, otázkou je, zda jsou tyto informace dostačující. Standardní spektrogram je zobrazením změn amplitudy signálu v čase, alternativní možností je však mel spektrogram. Mel spektrogram propisuje frekvence na melodickou stupnici a tím poskytuje lepší vhled do drobnějších slyšitelných nuancí audio stopy.

Mel spektrogram lze vytvořit modifikací původních spektrogramů pomocí následujícího nativního TensorFlow kódu.

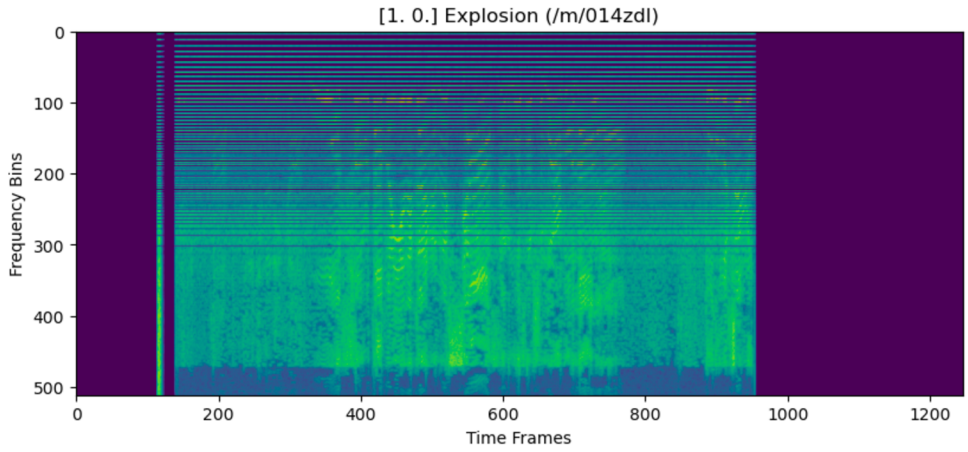
```
mel_weights = tf.signal.linear_to_mel_weight_matrix(N_MELS, spectrogram.shape[-1],
SAMPLE_RATE, 20, SAMPLE_RATE/2)
spectrogram = tf.tensordot(spectrogram, mel_weights, 1)
spectrogram = tf.math.log(spectrogram + 1e-6)
```

Příklad kódu 14 - Transformace spektrogramu na mel spektrogram - Zdroj: Autor

Výsledkem této transformace jsou následující spektrogramy.



Graf 9 - Mel spektrogram zvuku střelby - Zdroj: Autor



Graf 10 - Mel spektrogram zvuku exploze - Zdroj: Autor

Zásadním úkolem zastřešující funkce pro načtení dat je zajištění maximální vyváženosti dat. Toho je dosaženo dvěma kroky. Prvním krokem je možnost omezit množství načtených segmentů pro každý štítek. Funkce implementuje parametr limit, který může nabývat číselné hodnoty specificky omezující kolik segmentů za každý štítek bude maximálně načteno. Zároveň může parametr nabývat hodnoty v podobě řetězce „same“. Tato hodnota zajistí, že se pro každý štítek načte náhodně z celkové množiny tolik segmentů, kolik odpovídá četnosti nejméně zastoupeného segmentu.

```
sample_counts = metadata['positive_labels'].apply(lambda labels: any(label in labels for label in
target_classes.keys())).value_counts()
if limit == 'same':
    limit = sample_counts.min()
else:
    limit = int(limit)
metadata = metadata.groupby('positive_labels').apply(lambda x: x.sample(n=min(len(x), limit),
replace=True)).reset_index(drop=True)
```

Příklad kódu 15 - Omezení načtené množiny audio souborů pro každý štítek - Zdroj: Autor

Dalším opatřením zajišťujícím vyváženost datasetu je zajištění co možná nejlepšího promíchání segmentů z různými štítky napříč datasetem. Toho je docíleno tím, že je pro každý jednotlivý štítek je vytvořen zvláštní dataset, nad nímž jsou provedeny transformace na spektrogramy. Tyto separátní datasety jsou následně zkombinovány do jednoho pomocí funkce TensorFlow funkce sample_from_datasets, která zajistí rovnoměrné rozložení jednotlivých štítků.

```
datasets = []
# opakování pro každý štítek
for label in target_classes.keys():
    #vyfiltrování konkrétních segment s daným štítkem
    class_df = df[df['positive_labels'].apply(lambda x: label in x)]
    # transformace štítků
    class_df['encoded_labels'] = class_df['positive_labels'].apply(lambda x: encode_labels(x,
target_classes))
    #vytvoření datasetu
    class_dataset = tf.data.Dataset.from_tensor_slices((class_df['file_path'].values,
np.stack(class_df['encoded_labels'].values)))
    #transformace na spektrogramy
    class_dataset = class_dataset.map(process_file, num_parallel_calls=tf.data.AUTOTUNE)
    #vyfiltrování prázdných záznamů
    class_dataset = class_dataset.filter(filter_empty_labels)

    datasets.append(class_dataset)

#promíchání datasetů do jednoho
balanced_dataset = tf.data.Dataset.sample_from_datasets(datasets)
```

Příklad kódu 16 - Vyvážené načtení datasetu - Zdroj: Autor

4.6 Návrh a učení modelu

Nejkomplikovanější částí celého procesu zpracování úlohy byl návrh modelu. Na začátku tohoto procesu bylo nutné nastavit počáteční architekturu a parametry modelu, od kterých se následně odvíjela sekvence iterativních změn architektury a hyperparametrů. Cílem bylo nalézt takovou cílovou kombinaci, která povede k ideálnímu výsledku při trénování modelu a následné predikce modelu.

Ještě před tvorbou modelu však bylo nutné načíst do paměti potřebná trénovací a validační data. Tento úkon byl také předmětem iterativních změn, které postupně vedly k lepšímu výkonu při trénování modelu. Změny týkající se načtení dat byly prováděny zejména na tvaru a rozlišení výsledných spektrogramů, které produkuje funkce pro načtení dat ale také v omezení množiny klasifikovaných typů zvukových událostí.

Pro načtení dat posloužila funkce load_dataset, připravená v předchozím kroku. Dataset, který tato funkce vrací byl následně pro potřeby trénování a validace rozdělen na trénovací a validační subset v poměru 8:2. Načtení a rozdělení datasetu je vyobrazeno na následujícím příkladu.

```
#načtení datasetu
dataset = load_dataset(TARGET_CLASSES, 'dataset_filtered/train_all.csv', 'same')

#uložení celkové délky datasetu
dataset_len = len(dataset)
```

```
#výpočet délky trénovacího a evaluačního datasetu
train_dataset_len = int(dataset_len * 0.8)
eval_dataset_len = dataset_len - train_dataset_len

#zamíchání celého datasetu
dataset = dataset.shuffle(buffer_size=10000).cache()

#rozdělení datasetu na trénovací a evaluační
train_dataset = dataset.take(train_dataset_len)
eval_dataset = dataset.skip(train_dataset_len).take(eval_dataset_len)

#načtení datasetů do paměti, implementace opakování pro celý počet epoch
train_dataset =
train_dataset.batch(batch_size).repeat(epochs).prefetch(buffer_size=tf.data.AUTOTUNE)
eval_dataset = eval_dataset.batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)

#výpočet trénovacích a evaluačních kroků pro jednu epochu
steps_per_epoch = train_dataset_len // batch_size
validation_steps = eval_dataset_len // batch_size
```

Příklad kódu 17 - Rozdělení načteného dataasetu - Zdroj: Autor

Na příkladu je patrné použití tří dosud nespecifikovaných proměnných: TARGET_CLASSES, epochs a batch_size. První proměnná reprezentuje množinu typů zvukových událostí, které budeme klasifikovat, tato proměnná je objektem přiřazujícím název štítku danému identifikátoru štítku a je načtena z csv souboru popsaného v kapitole Příprava datasetu.

Druhou proměnnou je počet epoch, po které bude model hlubokého učení trénován na trénovacích datech. V průběhu učení modelu bylo s hodnotou počtu epoch experimentováno, než bylo dosaženo finální hodnoty. V počátcích trénování byl počet epoch z původní hodnoty 20, zvýšen postupně až na 100. Bylo ověřeno, že počet epoch nemá smysl omezovat nižší ani nijak vyšší hranicí, ale spíše je užitečné nad modelem implementovat mechanismus předčasného ukončení trénování. V praxi modely s předčasným ukončením, s kterými bylo experimentováno téměř nikdy nedosáhly hranice 100 epoch. Modely v průběhu trénování efektivně snížily svou validační ztrátu mezi epochami 40-50 pod 10 % a následně dosáhly ideální ztráty při validaci mezi 50-70 epochou v extrémních případech, kdy byla trpělivost mechanismu pro ukončení trénování zvýšena nebo v případě, kdy byla ranným modelům předložena více nevyvážená data, dosahoval model ideální ztráty ukončující trénink později mezi 60-80 epochou. Pozdější modely potom automaticky ukončovaly učení z důvodu dosažená ideální ztráty mezi 50-60 epochou. Počet epoch byl tedy ve finále nastaven na hodnotu 60, která byla vyhodnocena jako optimální po finální typ použitého modelu, který bude popsán dále v textu.

Ruku v ruce s počtem epoch bylo experimentováno s velikostí dávek. U této proměnné bylo u všech typů modelů vyhodnoceno, že větší dávky zásadně snižují výkon modelu při

trénování. Z původní hodnoty 16 byla velikost dávky postupně experimentálně zvyšována až na hodnotu 32. Takováto hodnota vzhledem k velikosti celého datasetu nepřinášela dobré výsledky a vedla dokonce k rostoucí ztrátě v pozdějších epochách trénování, a tedy i k automatickému ukončení trénování předčasně i při zvýšené trpělivosti automatického mechanismu ukončení trénování. Zároveň se v experimentech neprojevilo pozitivně ani snížení velikosti dávky pod hodnotu 16. Z toho důvodu byla ponechána na původní hodnotě.

Při návrhu modelu hlubokého učení pro tuto úlohu bylo přikročeno k použití konvolučních vrstev. Spektrogramy, které generuje funkce pro načtení dat jsou v zásadě 2D obrazovou informací reprezentující dané audio záznamy. Díky tomu lze na spektrogramy aplikovat 2D konvoluční vrstvy, které z nich efektivně extrahují potřebné vzorce. Vzhledem k charakteru trénovacích dat, je potom na místě použití minimálně dvou navazujících konvolučních vrstev pro posílení této schopnosti a co nejlepší extrakce vzorců v datech.

Pro lepší ilustraci výchozího bodu, od kterého se odvíjely veškeré úpravy je zde uveden příklad původního modelu.

```
model = Sequential([
    Input(input_shape=input_shape)
    Conv2D(16, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

model.compile(
    'Adam'
    loss='categorical_crossentropy',
    metrics=[
      tf.keras.metrics.Accuracy(name='accuracy'),
      tf.keras.metrics.Recall(name='recall'),
      tf.keras.metrics.Precision(name='precision'),
      MacroF1Score(num_classes= num_classes)
    ]
)
```

Příklad kódu 18 - Výchozí architektura modelu - Zdroj: Autor

Vstupní vrstva modelu je neměnným prvkem s definovaným tvarem vstupního tensoru. V průběhu trénování různých modelů bylo však s tvarem vstupu experimentováno. Tvar vstupu je jasně definován tvarem spektrogramu. V první fázi testování byly modely trénovány na spektrogramech ve tvaru (622, 256, 1). Tento tvar byl pomocí změn parametrů funkce pro načtení dat postupně upravován. Bylo experimentováno s nižším rozlišením

spektrogramu, avšak výkon při trénování nad těmito spektrogramy byl nedostačující tíhnoucí k vysoké validační ztrátě a nízké úrovni rozpoznání navazujících příkladů. Proto byla funkce upravena, aby generovala spektrogramy ve vyšším rozlišení. Nutné bylo najít ideální velikost spektrogramu z níž model dokáže informace získat ale zároveň takovou velikost, která bude znamenat přijatelné technologické nároky zejména na paměť. Nakonec bylo postupnou iterací dosaženo nejpříznivějších výsledků se spektrogramy ve tvaru (1247, 512, 1). Tento tvar nabízí modelu dostatečné rozlišení pro extrakci menších nuancí a zároveň znamená přijatelné nároky na paměť a rychlost učení.

Výstupní vrstva je taktéž neměnným prvkem, avšak i počet výstupů se v průběhu trénování měnil. Z původní množiny kategorií audio záznamů byly postupně vyřezovány ty, u nichž predikce nedosahovaly zdaleka dostatečné úrovně přesnosti. Mezi těmito třídami zvuků byly zvuky ohně, a tříštění skla. Vyřazení těchto tříd podporuje i skutečnost, že i v rámci celého datasetu dosahují maximálně kvalitativní úrovně 60 % jak je uvedeno na webu AudioSetu.

Důležitým parametrem výstupní vrstvy je její aktivační funkce. Vzhledem k povaze úlohy byla zvolena aktivační funkce softmax. Tato funkce zajišťuje výpočet pravděpodobnosti výstupu pro každý výstupní neuron zvlášť a hodí se spíše pro datasety, kde se vyskytuje více štítků pro jeden záznam. Tuto vlastnost AudioSet má, avšak jak již bylo zmíněno, záznamy s vícero štítky byly při přípravě datasetu vyfiltrovány pro svou velmi nízkou četnost. V průběhu trénování bylo experimentováno s použitím aktivační funkce softmax i sigmoid a bylo odhaleno, že použití softmax funkce vede k lepším výsledkům při trénování i predikci. V konečných fázích experimentování bylo upuštěno of vyfiltrování záznamů s více štítky, což nepřineslo žádné zhoršení trénování modelu, naopak je pravděpodobné, že každý další použitý záznam při trénování přispěl k pozitivnějšímu výsledku trénování vhledem k malé velikosti datasetu.

Ruku v ruce s aktivační funkcí výstupní vrstvy bylo experimentováno s typem ztrátové funkce. Funkce categorical crossentropy byla na začátku zvolena jako ideální v situaci, kdy je pro výstupní vrstvu použita aktivační funkce softmax. Při změně na aktivační funkci sigmoid byla změněna i ztrátová funkce na binary crossentropy. V prvotních fázích trénování byla tato kombinace efektivní při nalezení ideální architektury modelu, avšak velmi brzy se ukázaly její limity. Kombinace funkcí dosahovala ideálních stavů metrik recall a precision velmi brzy a vedla k přeučení. Původní kombinace softmax a categorical

crossentropy se ukázala jako lepší při trénovacím procesu i následné predikci, neboť dosáhla pozvolnějšího vylepšování všech metrik.

Metrikami sledovanými v trénovacím procesu byly recall neboli míra rozpoznání vstupu, precision neboli přesnost pozitivních výsledků modelu, accuracy jakožto obecná míra přesnosti určení výsledku a samozřejmě loss neboli ztráta. Metriky byly sledovány zejména ve více vypovídající validační fázi, která se v počátečních fázích trénování od trénovací odchylovala svými výsledky, avšak lepším promícháním datasetu bylo dospěno k lepším výsledkům, při nichž validační metriky lépe kopírovaly ty trénovací. V některých fázích trénování byla navíc použita metrika MacroF1Score. Tato metrika je průměrem metrik F1Score pro každou třídu klasifikace. F1Score je pak harmonickým průměrem metrik recall a precision. Díky této metrice lze lépe sledovat, jak trénování modelu funguje s datasetem tíhnoucím k nevyváženosti a je lepší reprezentací úspěšnosti modelu než metriky recall a precision samy o sobě.

Ačkoli byla pro trénování použita metrika přesnosti (accuracy) je důležité zmínit, že pro daný dataset je tato metrika často zavádějící. Dataset totiž vykazuje velkou nevyváženost, a to ne nutně početní ale spíše nevyváženost v doméně charakteru samotných dat a jejich podoby. Proto i přesto, že se podařilo trénovací data vyvážit tak, že každý štítek byl zastoupen relativně vyrovnaným množstvím záznamů, stále v datasetu přetrvala vysoká variabilita a nelinearita dat.

Proces experimentování s různými architekturami nakonec vedl k několika úpravám modelu. Experimentování bylo většinově procesem iterativních změn a sledování výsledků trénování a predikce. Tento proces byl velice časově náročný, v jeho průběhu bylo vytvořeno několik desítek modelů, přičemž většina modelů byla trénována s několika verzemi způsobů načtení, filtrace a promíchání dat. Finální architektura modelu, která přinesla nejlepší výsledky je vyobrazena na následujícím příkladu.

```
model = Sequential([
    Input(shape=input_shape),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
```

```
model.compile(
    'rmsprop',
    loss='categorical_crossentropy',
    metrics=[
      tf.keras.metrics.Accuracy(name='accuracy'),
      tf.keras.metrics.Recall(name='recall'),
      tf.keras.metrics.Precision(name='precision'),
      MacroF1Score(num_classes=num_classes)
    ]
)
```

Příklad kódu 19 - Architektura finálního nejúspěšnějšího modelu - Zdroj: Autor

Jak je patrné modelu zůstaly jeho dvě konvoluční vrstvy extrahující detaily ze vstupu. Dále zůstala přítomna jedna Dense vrstva nově se 128 neurony. Na rozdíl od původního návrhu však nový model má více filtrů v konvolučních vrstvách. Tato úprava vyplynula z pokusu o maximalizaci validační přesnosti (val_accuracy). Hledání maximální přesnosti probíhalo pomocí knihovny kerastuner. Tato knihovna umožňuje provést nad modelem náhodné hledání hodnot hyperparametrů (kerastuner.RandomSearch). Předmětem hledání byly právě i filtry konvolučních vrstev u nichž se po několika kolech testovacího trénování projevilo, že model dosahuje vyšší přesnosti s větším počtem filtrů. Reálné hodnoty, ke kterým proces náhodného hledání dospěl (96 filtrů) však nebyly ideální, neboť kerastuner.RandomSearch lze zacílit na vylepšení pouze jedné metriky, nikoliv na vyvážení několika metrik najednou. Náhodné hledání však pomohlo určit směr, kterým se při úpravách modelu vydat. Proto bylo snazší iterativními úpravami filtrů nakonec dospět k finální hodnotě 64 filtrů pro obě konvoluční vrstvy. Tato konfigurace poskytla modelu nejlepší výsledky při trénování ze všech vyzkoušených architektur modelů. Předmětem experimentování bylo i přidání jedné nebo dvou konvolučních vrstev navíc. Modely s touto konfigurací však nevykazovaly dobré výsledky a tíhly k vyšší ztrátě. Dalším doplňkem konvolučních vrstev je 25 % droupout, který pozitivně přispěl k pozvolnějšímu dosažení ideálních hodnot metrik recall a precision.

Experimentováno bylo i s přidáním vrstvy Normalization před konvoluční vrstvy, které pro některé trénovací cykly přinesla pozitivní výsledek při trénování, avšak s modifikací dat v konečných fázích testování se tento efekt vytratil a odebrání vrstvy nakonec přineslo pozitivnější výsledky. Stejně tak nebylo přínosné ani přidání vrstvy BatchNormalization ke konvolučním vrstvám. Tato modifikace vedla k vyšší ztrátě. Další ověřovanou změnou architektury bylo přidání více Dense vrstev a různá konfigurace počtu neuronů. Pozitivní efekt měla také změna optimalizačního algoritmu z Adam na RMSprop. Na mnoha

trénovacích cyklech se stejnou architekturou po modifikaci optimalizačního algoritmu vykazoval model lepší výsledky, co se týče ztráty s algoritmem RMSprop.

Trénování modelů bylo až na několik prvotních trénovacích cyklů spouštěno s callback mechanismy brzkého zastavení a změny rychlosti učení.

```
callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.0001)
]

history = model.fit(
    train,
    epochs=EPOCHS,
    validation_data=test,
    verbose=1,
    steps_per_epoch=train_steps,
    validation_steps=test_steps,
    callbacks=callbacks
)
```

Příklad kódu 20 - Spuštění trénování modelu - Zdroj: Autor

Tyto mechanismy posloužily jako efektivní nástroj proti přeučení modelu. V momentě dosažení bodu, kdy se ztráta modelu nijak významně nezlepšila zajistil mechanismus ReduceLROnPlateau snížení rychlostí čímž bylo docíleno podrobnějších změn vah a model se tak precizněji dopracoval lepší přesnosti a mnohdy lepší ztráty. V tandemu mechanismus EarlyStoping zajistil případné ukončení trénování, pokud se ztráta modelu nezlepšila ani v širším pohledu. Nastavení trpělivosti pro tento callback mechanismus je proto větší než pro mechanismus změny rychlosti učení. Oba mechanismy se ukázaly být velmi efektivní a projevily svou účinnost v každém trénovacím procesu, kde byly implementovány.

Jak již bylo zmíněno AudioSet se skládá ze dvou subsetů: slabě anotovaného a silně anotovaného. Oba subsety se navíc dělí na trénovací a validační subset, přičemž slabě anotovaný trénovací subset se dále dělí na vyvážený trénovací a nevyvážený trénovací subset. Původním konceptem strategie trénování bylo použít vyvážený slabě anotovaný subset na natrénování modelu, následně modelu snížit rychlost učení, případně zmrazit některé vrstvy a následně provést dotrénování modelu na silně anotovaném subsetu. Tato strategie byla v mnoha pokusech implementována, avšak nedosáhla uspokojivých výsledků.

Součástí každé nasazené trénovací strategie bylo případné zmenšení množiny klasifikovaných zvukových událostí. U každé strategie se tak postupně původní kompletní klasifikační množina zmenšovala na nejmenší přípustnou množinu obsahující tři třídy zvuků: ticho, křik a střelbu. Tato redukovaná úloha byla snahou o ověření funkčnosti

datasetu pro trénování pro danou úlohu s cílem eliminovat nepotřebné třídy a stále dosáhnout minimální užitečnosti výsledného modelu.

Jak již bylo zmíněno, původní strategií tedy bylo trénovat modely pouze na vyvážených slabě anotovaných datech a následně je dotrénovat na silně anotovaných datech. Tato strategie se rychle ukázala jako nedostačující, protože obsahovala příliš málo záznamů, proto byla k vyvážené slabě anotované množině přidána ještě nevyvážená slabě anotovaná množina. Tím bylo navýšeno celkové množství záznamů pro trénování a zároveň byla datům dodána větší míra variability. Zároveň bylo nutné data důsledně promíchat, aby se různé typy dat rovnoměrně rozmělnily napříč datasetem. Tento směr se ukázal jako přínosný, a proto bylo přikročeno k dalšímu kroku: přidání separátní evaluační množiny do trénovacích dat. Tato množina byla sice autory datasetu zamýšlena jako čistě evaluační, avšak v kontextu dané úlohy, která nepoužívá kompletní AudioSet ale pouze jeho subset, se ukázalo zahrnutí této množiny do trénovacích dat jako přínosné. Evaluační data přinesla opět větší míru variability a nelinearity, z které se model nyní dokázal lépe učit skryté vzorce.

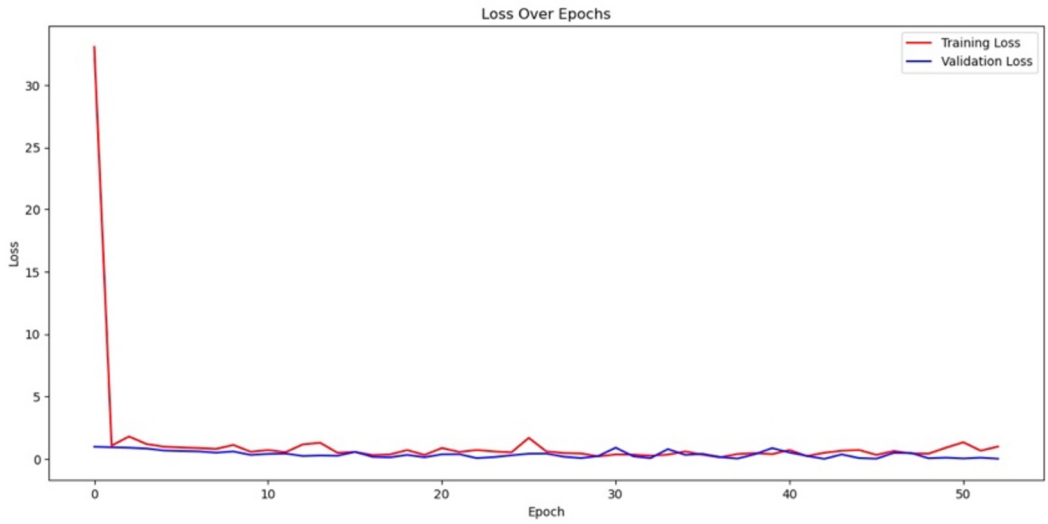
Strategie však měla zásadní problém spočívající v následném dotrénování. Tento krok se ukázal jako velmi kontraproduktivní. Po zdlouhavém vyvažování silně anotované množiny stále nebylo dosaženo přijatelných výsledků po dotrénování modelu. Tato operace model přiměla zcela přestat vnímat některé třídy a jiné naopak chybně vyhodnocoval jako falešně pozitivní. Dotrénování silně anotovanou množinou bylo v pozdějších fázích proto zavrženo jako slepá cesta.

Model trénovaný pouze na slabě anotované množině však stále nedosahoval zdaleka požadovaných výsledků při predikci. Poslední trénovací strategií, která byla pro modely implementována zahrnovala zkombinování původní slabě anotované množiny vyvážených, nevyvážených a evaluačních dat se silně anotovanou trénovací a evaluační množinou v jednom trénovacím cyklu. Tato strategie se nakonec ukázala jako nejefektivnější v přiblížení požadované přesnosti modelu při predikci. Kombinace všech typů dat v jednom trénovacím cyklu přinesla modelu dostatečně variabilní datovou sadu a výsledný model predikoval výsledky lépe než modely původně trénované na slabě anotované množině a dotrénované na silně anotované množině.

Výsledky trénování modelů se postupně dařilo vylepšovat a některé metriky se podařilo opakovaně vylepšovat bez větších problémů. Kde však nastal problém, byla přesnost modelu

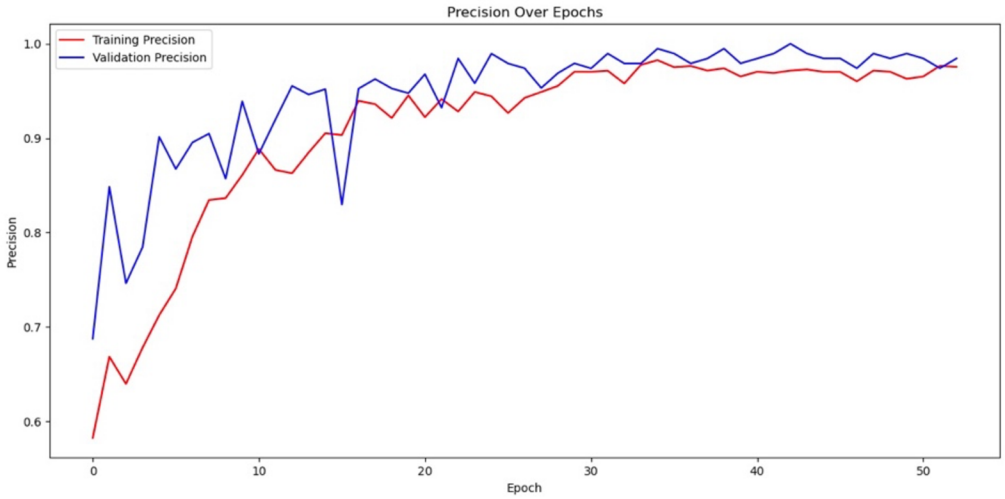
(accuracy). Tuto metriku se dlouhodobě nedařilo vylepšovat, predikce většiny modelů byla vzdálena uspokojivému stavu a blížila se nanejvýše úrovni 40 %. Predikce byla u modelů ověřována nad validačním subsetem a její výsledky byly vysoce proměnlivé vzhledem k načtení validačních dat.

Jeden z modelů tuto hranici překročil, avšak jeho predikce stále zůstala velmi proměnlivá a nestabilní. Výsledek trénování tohoto modelu je vyobrazen na následujících grafech. Podobných výsledků dosahovalo v pozdější fázi více modelů, avšak u všech platilo že ačkoli byly výsledky trénování poměrně uspokojivé, výsledky predikce byly daleko od uspokojivého stavu. Ztrátu modelů se téměř vždy podařilo srazit na ideální úroveň. U starších modelů trénovaných pouze na slabě anotovaném datasetu byl sestup plynulejší, na grafu č. 11 je však vidět efekt promíchání několika typů subsetů do jednoho projevující se občasnými nárůsty ztráty a následnými pády. Obecně však trajektorie ztráty směřovala k uspokojivému výsledku trénování.

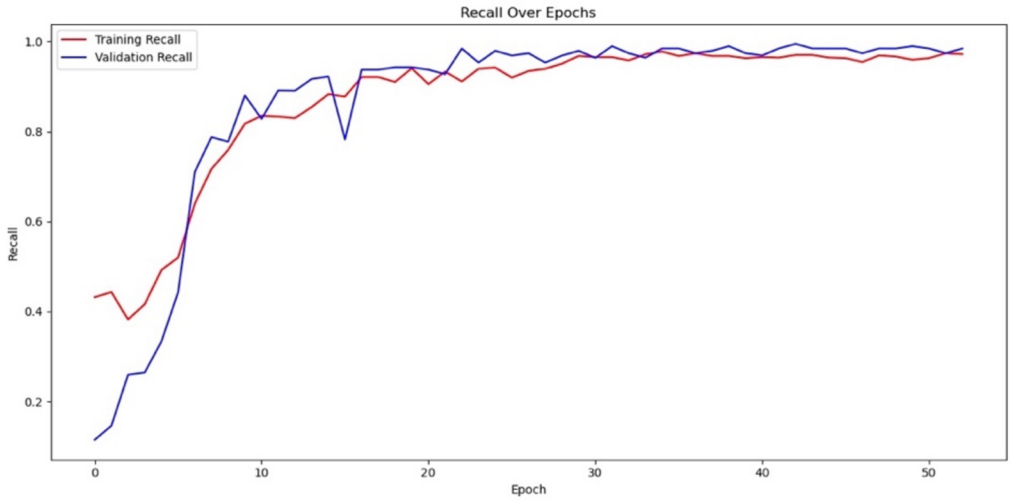


Graf 11 - Ztráta výsledného modelu - Zdroj: Autor

Metriky recall a precision se u většiny modelů vyvíjely velmi podobně. Na grafech č. 12 a 13 je opět vidět efekt promíchání různých typů dat, avšak typický strmý vzestup v ranných epochách a následné zbrzdění růstu bylo charakteristické pro téměř všechny trénované modely.

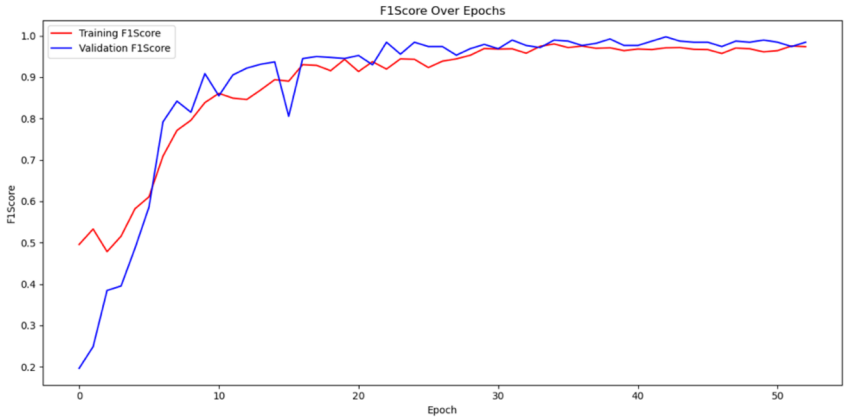


Graf 12 - Metrika precision výsledného modelu - Zdroj: Autor



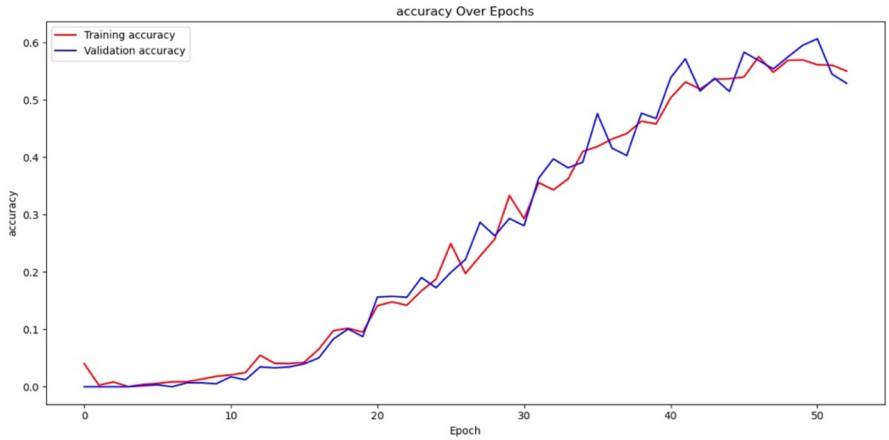
Graf 13 - Metrika recall výsledného modelu - Zdroj: Autor

Metrika F1Score reprezentuje vyvážení mezi metrikami recall a precision. Tato metrika je užitečnější než recall a precision ve vyjádření úspěšnosti trénování modelu s nevyváženým datasetem, což bylo v pozdějších fázích trénování pravidlem. V tomto případě (graf č. 14) je vidět, že vyvážení metrik je na dobré úrovni a trénování modelu je tak možné považovat za úspěšné.



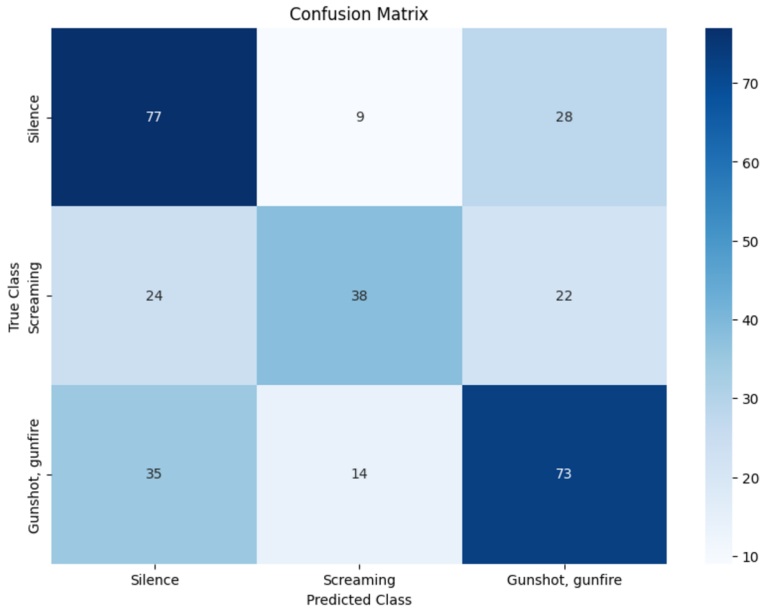
Graf 14 - Metrika F1Score výsledného modelu - Zdroj: Autor

Poslední sledovanou metrikou byla přesnost modelu, která by měla vyjadřovat možnou úroveň predikce modelu. Tato metrika u většiny modelů dosahovala velmi nízké úrovně. U některých modelů se přesnost nevyšplhala nad 10 %. Vzhledem k nevyváženosti některých testovacích dat nebyl metrice dáván větší význam, neboť v takovém kontextu má tendenci být velmi nepřesná. A vskutku nepřesnost této metriky byla opakovaně ověřena na mnoha modelech. U některých modelů jejichž přesnost při trénování nedosahovala ani 10 % nakonec přesnost predikce překročila i 20 %. Naopak u modelů, u nichž přesnost vystoupala na 40 % nebo více nakonec predikce odhalila úspěšnost nižší.

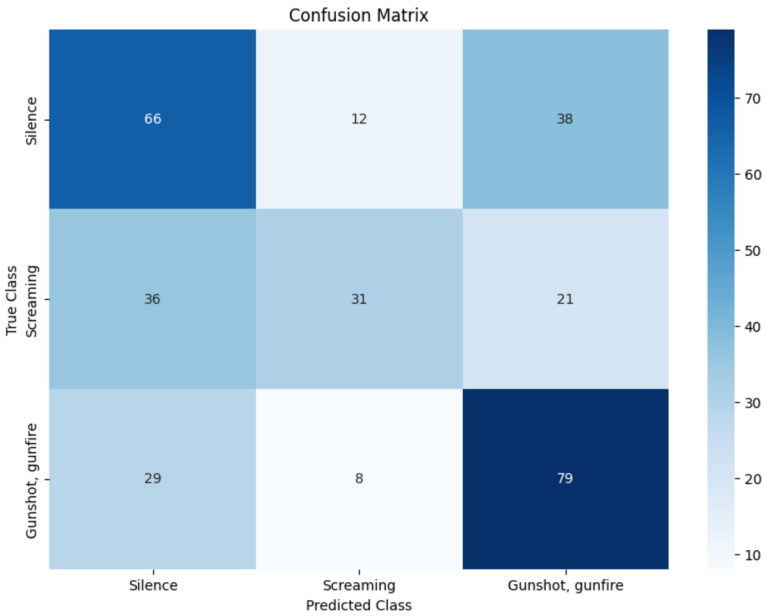


Graf 15 - Přesnost výsledného modelu - Zdroj: Autor

Pro ověření úspěšnosti modelu byla po provedení predikcí sestavena matice záměn. Tato metoda porovnává skutečné hodnoty validační množiny s výstupy modelu a podobnosti i rozdíly přehledně zobrazuje v tabulce. Bohužel u většiny modelů se nepodařilo dosáhnout přijatelných výsledků predikce. Příklady výsledků nejúspěšnějšího modelu, jsou vyobrazeny na následujících maticích záměn.



Graf 16 - Matice záměn výsledného modelu 1 - Zdroj: Autor



Graf 17 - Matice záměn výsledného modelu 2 - Zdroj: Autor

Na maticích lze pozorovat že model nejlépe detekoval přítomnost třídy ticho a střelba, avšak stále predikoval velké množství falešně pozitivních a falešně negativních výsledků. U úlohy jako je tato, je tento výsledek neuspokojivý, neboť by v reálném použití vedl k planým poplachům a v některých případech ignorování nebezpečné situace.

Obecně se dá říct, že modely byly i přes svou zjevnou úspěšnost při trénování nakonec velmi nespolehlivé v klasifikaci. Za takto neúspěšným chováním modelů je několik faktorů, které budou specifikovány v následující kapitole.

5 Výsledky a diskuse

Úloha zpracovávaná v rámci této práce měla za cíl prověřit možnost vytrénování nové hluboké neuronové sítě nad datasetem AudioSet pro klasifikaci audio záznamů. Tento dataset samozřejmě již byl v minulosti použit k trénování jiných neuronových sítí jinými autory a týmy včetně autorů samotného AudioSetu, tedy týmem zabývajícím se strojovým porozuměním zvukovým událostem pod firmou Google. Tito autoři přistupovali k datasetu jako k nedělitelnému celku a výsledné modely tak jsou robustními nástroji trénovanými na stovkách zvukových kategorií a milionech audio záznamů. Vývoj takových modelů je velmi finančně i technicky nákladný a vyžaduje využití robustní výpočetní a paměťové infrastruktury. AudioSet však obsahuje množství tematicky provázaných tříd, které otevírají teoretickou možnost natrénovat nad vybraným subsetem specializovanější odlehčený model.

Jelikož tedy vytrénování plného modelu hlubokého učení nad kompletním datasetem nebylo technicky možné, bylo v rámci této práce z datasetu vyčleněno několik kategorií zvuků, které byly pro k trénování modelu zužitkovány. Výběrem specifických kategorií vznikl subset čítající několik jednotek tisíc audio záznamů, na kterých mohl být model hlubokého učení trénován s realistickým výhledem i v omezených technických podmínkách. Zvukové události byly vybrány tak aby tematicky spadaly do kategorie zvuků představujících případné ohrožení lidí nebo majetku. Takovými zvuky jsou zejména křik, střelba, exploze, tříštění skla a další.

V rámci zpracování úlohy byl nejprve naprogramován algoritmus, umožňující stažení použitelných dat popsaných datasetem. Tento algoritmus plně respektuje technologické limitace a je optimalizován tak aby dokázal získat potřebná data v nejlepší možné kvalitě co nejefektivněji, neboť je sestrojen na míru dané úloze.

Pro zpracování stažených dat byl následně navržen algoritmus pro jejich načtení a transformaci do podoby spektrogramů. Tento algoritmus záměrně deleguje některé transformační operace algoritmu pro stažení dat, proto aby tyto operace nebylo nutné vykonávat opakovaně vždy při načtení dat. Díky tomu je načtení dat pomocí tohoto algoritmu optimalizováno pro spouštění v technicky limitovaném prostředí, avšak zachovává si možnost parametricky upravit výstup algoritmu pro prostředí technicky vyspělejší. Zároveň je algoritmus navržen tak aby načtená data byla co nejvíce vyvážená pro potřeby trénování modelu s menším datasetem.

Nakonec byl v rámci práce trénován model hlubokého učení, který by dokázal vybrané kategorie zvuků klasifikovat. Proces návrhu architektury modelu byl velmi zdlouhavý, neboť zahrnoval hledání vhodné kombinace vstupních parametrů modelu a dat, které by vedly k uspokojivému výsledku. Za celou dobu experimentování bylo natrénováno několik desítek modelů, u kterých se postupně dařilo vylepšovat výkon v procesu učení sledováním klíčových metrik a iterativním laděním hyperparametrů. Důležité metriky se podařilo při trénování úspěšně vylepšovat, avšak modely si obecně nevedly dobře v predikci. Nepodařilo se tedy naplnit jejich potenciál v úloze klasifikace.

Ačkoli modely prošly trénovacím procesem velmi úspěšně, žádný z nich nedospěl uspokojivých výsledků v predikci výsledků nad validační množinou dat. To znamená, že se nedokázaly naučit z dostupných dat dostatečně generalizovat. Tento problém, lze připsat nedostatku trénovacích dat, a vskutku vstupních trénovacích dat nebylo k dispozici tolik, vzhledem k tomu, že z kompletního datasetu byla pro trénování vybrána jen malá množina kategorií zvuků. Přesto však bylo ověřeno, že lepších výsledků paradoxně dosahují modely trénované na nejmenší množině kategorií zvuků. Problém tedy leží trochu jinde.

Při analýze kompletního datasetu byla zjištěna jedna jeho důležitá vlastnost, a to sice vysoká míra provázání některých kategorií zvuků v jednotlivých záznamech. Většina slabě anotovaných audio záznamů popsaných datasetem totiž v základu obsahuje ve velké míře dvě a více kategorií zvuků. Při výběru subsetu pro trénování z kompletního datasetu však byly informace o nepotřebných kategoriích z vybraných záznamů odebrány a výsledný subset rázem vykazoval přesně opačnou vlastnost, tedy že většina audio záznamů v subsetu obsahuje jednu kategorii. Tato skutečnost implikuje velkou ztrátu vnějšího kontextu pro každý slabě anotovaný audio záznam. Tento kontext by dokázal zužitkovat velkorozměrný model trénovaný na kompletním datasetu

Zjištění provedená v rámci této práce ukazují na to, že AudioSet není příliš vhodným nástrojem pro trénování modelu klasifikujícího malý výsek kategorií z daného datasetu. Takový výsek (subset) není dostatečně specializovaný na to, aby vykazoval dobré výsledky a zároveň neobsahuje dostatečný kontext, který nabízí plný dataset.

Zjištění tedy směřují ke dvěma alternativním řešením, ke kterým by s nově nabytými informacemi bylo možné přikročit spíše než k řešení, jaké bylo v rámci této práce nad úlohou aplikováno. Prvním řešením by bylo přistupovat k datasetu jako k celku, jak činí samotní

jeho autoři. Možnou cestou by bylo natrénovat velkorozměrný model nad celým datasetem nebo většinou jeho nejprominentnějších zvukových kategorií. Tato cesta by však vyžadovala řádově lepší technické zázemí. Proto by bylo vhodnější dotrénovat již existující model, který nad datasetem byl trénován v minulosti pomocí metody přenosového učení. Tato cesta by jistě vedla k spolehlivějšímu výslednému modelu, který by se lépe osvědčil v situacích, v nichž není příliš prostoru pro omyl jako tomu bylo v případě úlohy řešené v rámci této práce.

Druhým alternativním řešením by bylo trénovat model pouze nad jednou nebo dvěma nejprominentnějšími kategoriemi zvuků z AudioSetu. Takový model by nebyl určen pro úlohy, kde je důležitá preciznost ale dobře by posloužil pro nenáročné úlohy rozpoznání jednotlivých zvukových událostí, vyžadujících minimalistický model strojového učení.

AudioSet je obecně velmi rozsáhlý a obsahuje velké množství kategorií zvuků, které nejsou napříč audio záznamy zastoupeny rovnoměrně. Nabízí tedy data vhodná pro různé úlohy, avšak pro každou takovou úlohu je důležité dobře analyzovat, zda se hodí provést z datasetu výběr subsetu, nebo je vhodnější dotrénovat již existující model trénovaný nad celým AudioSetem nebo jiným audio datasetem.

6 Závěr

Tato diplomová práce se zabývala vývojem hluboké neuronové sítě pro úlohu klasifikace audio záznamů z velkorozměrné datové sady AudioSet. Cílem práce bylo ověřit možnost vytrénování menšího specializovaného modelu hlubokého učení, který by rozpoznával specifické kategorie zvuků znamenajících potenciální ohrožení osob nebo majetku. V rámci práce byla sestavena rešerše na téma hlubokého učení, neuronových sítí a nástrojů pro jejich realizaci, byly rozebrány metodiky trénování neuronových sítí a užitečné praktické postupy aplikovatelné v iterativním procesu trénování pro dosažení co nejlepších výsledků. Dále byla provedena analýza použitého datasetu a výběr subsetu pro zvolenou úlohu. Na míru řešené úloze byl naprogramován optimální algoritmus pro stažení dat popsaných datasetem. Pro zpracování stažených dat byl naprogramován další algoritmus, který data transformuje do podoby spektrogramu, která je zpracovatelná hlubokou neuronovou sítí. Nad transformovanými daty bylo následně trénováno několik desítek modelů hlubokého učení v iterativním procesu experimentování s různými architekturami modelů a různými konfiguracemi vstupních dat.

Navzdory původnímu záměru se v průběhu experimentování s různými typy modelů vyskytly komplikace, v reakci na něž bylo nutné učinit několik kompromisů. Bohužel i přes zavedené kompromisy nebylo možné dosáhnout modelu, který by uspokojivě plnil zvolenou úlohu. Díky teoretickým poznatkům z rešerše bylo sice dosaženo optimálního trénování modelu, avšak výsledný model nepředvedl dostatečnou schopnost klasifikace. Tento neúspěch lze přičítat tomu, že při selekci specifického subsetu z plného datasetu došlo ke ztrátě potřebných informací. Tento závěr vychází z faktu, že audio záznamy datasetu z většiny obsahují více kategorií zvuků najednou a výběrem podmnožiny dat přichází data o důležitý kontext. Dalším zjištěním práce ale je i to, že modely trénované na datasetu nabývají lepších výsledků úměrně se zmenšováním množiny klasifikovaných zvukových kategorií.

Závěrem práce je proto to, že použití AudioSetu je vhodnější buď pro dotrénování předpřipravených modelů pomocí metody přenosového učení nebo trénování nového modelu s podmínkou využití kompletního datasetu nebo jeho většinové části. Zároveň by ale AudioSet mohl posloužit k trénování malých modelů pro rozpoznání jedné nebo maximálně dvou zvukových kategorií pro úlohy při nichž není důležitou podmínkou stoprocentní preciznost výstupu.

7 Seznam použitých zdrojů

ALTO, Valentina, 2019. Neural Networks: parameters, hyperparameters and optimization strategies. *Medium* [online]. [cit. 2024-03-27]. Dostupné z: https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5

BENTO, Carolina, 2021. Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. In: *Towards Data Science* [online]. [cit. 2024-03-31]. Dostupné z: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141

BROWNLEE, Jason, 2019. A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. *Machine Learning Mastery* [online]. [cit. 2024-03-28]. Dostupné z: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

BROWNLEE, Jason, 2019. A Gentle Introduction to Transfer Learning for Deep Learning. *Machine Learning Mastery* [online]. [cit. 2024-03-27]. Dostupné z: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

BROWNLEE, Jason, 2019. How to Configure the Number of Layers and Nodes in a Neural Network. *Machine Learning Mastery* [online]. [cit. 2024-03-27]. Dostupné z: https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

BROWNLEE, Jason, 2019. What is the Difference Between a Parameter and a Hyperparameter? *Machine Learning Mastery* [online]. [cit. 2024-03-27]. Dostupné z: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/

BROWNLEE, Jason, 2021. Weight Initialization for Deep Learning Neural Networks. *Machine Learning Mastery* [online]. [cit. 2024-03-27]. Dostupné z: https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/

BROWNLEE, Jason, 2022. Difference Between a Batch and an Epoch in a Neural Network. *Machine Learning Mastery* [online]. [cit. 2024-03-28]. Dostupné z: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

CHOLLET, François, [2018]. *Deep learning with Python*. Shelter Island, NY: Manning. ISBN 978-161-7294-433.

CHOLLET, Francois, 2015. About Keras 3. *Keras* [online]. [cit. 2024-03-28]. Dostupné z: https://keras.io/about/

CHOWDHURY, Prasenjit, 2023. Perceptron: A Simple yet Mighty Machine Learning Algorithm. In: *Medium* [online]. [cit. 2024-03-31]. Dostupné z: https://medium.com/@cprasenjit32/perceptron-a-simple-yet-mighty-machine-learning-algorithm-9ff6b7d86a71

DONGES, Nikolas a Jessica POWERS, 2023. Gradient Descent in Machine Learning: A Basic Introduction. *BuiltIn* [online]. [cit. 2024-03-27]. Dostupné z: https://builtin.com/data-science/gradient-descent

DOSHI, Ketan, 2021. Audio Deep Learning Made Simple: Sound Classification, Step-by-Step. *Medium* [online]. [cit. 2024-03-27]. Dostupné z: https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5

GÉRON, Aurélien, [2017]. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Beijing: O'Reilly. ISBN 978-1-491-96229-9.

GOOGLE LLC, 2015. TensorFlow basics. ABADI, Martin, Ashish AGARWAL, Paul BARHAM, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems* [online]. [cit. 2024-03-28]. Dostupné z: https://www.tensorflow.org/guide/basics

GOOGLE LLC, 2017. AudioSet. In: GOOGLE LLC. *AudioSet: A large-scale dataset of manually annotated audio events* [online]. [cit. 2024-03-31]. Dostupné z: https://research.google.com/audioset/

HAMMEL, B. D., 2019. What learning rate should I use? *B. D. Hammel* [online]. [cit. 2024-03-28]. Dostupné z: http://www.bdhammel.com/learning-rates/

HAYKIN, Simon S., 2009. *Neural networks and learning machines*. 3rd ed. Upper Saddle River: Pearson Education. ISBN 978-0-13-147139-9.

HOLČÍK, Jiří Holčík a Martin KOMENDA, 2015. *Matematická biologie: e-learningová učebnice* [online]. Brno: Masarykova univerzita [cit. 2024-03-31]. ISBN 978-80-210-8095-9. Dostupné z: https://portal.matematickabiologie.cz

KWIATKOWSKI, Robert, 2021. Gradient Descent Algorithm — a deep dive. *Towards Data Science* [online]. [cit. 2024-03-27]. Dostupné z: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

MOORE, Gordon, 1965. Cramming more components onto integrated circuits. *Electronics*. **38**(8).

Neuron, 2006. In: *Wikipedia Commons* [online]. [cit. 2024-03-31]. Dostupné z: https://cs.m.wikibooks.org/wiki/Soubor:Neuron_(cesky)-1.svg

NG, Andrew, 2023. Xavier Initialization and Regularization. STANFORD UNIVERSITY. *CS230 Deep Learning* [online]. [cit. 2024-03-27]. Dostupné z: https://cs230.stanford.edu/section/4/

NWANKPA, Chigozie Enyinna, Winifred IJOMAH, Anthony GACHAGAN a Stephen MARSHALL, 2020. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*.

PARENTE, Daniel, 2023. Fine-Tuning vs Training From Scratch Deep Learning Models: Which is Better? PARENTE, Daniel. *Danielparente.net* [online]. [cit. 2024-03-31]. Dostupné z: https://www.danielparente.net/en/2023/10/17/fine-tuning-vs-training-from-scratch-deep-learning-models/

PRAMODITHA, Rukshan, 2022. Parameters Vs Hyperparameters: What is the difference? *Medium* [online]. [cit. 2024-03-27]. Dostupné z: https://medium.com/data-science-365/parameters-vs-hyperparameters-what-is-the-difference-5f40e16e2e82

RADHAKRISHNAN, Pranoy, 2017. What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network? *Medium* [online]. [cit. 2024-03-28]. Dostupné z: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

RAI, Prashant, 2017. Introducing Gradient Descent : minimizing the cost function. In: *Medium* [online]. [cit. 2024-03-31]. Dostupné z:

https://medium.com/@prashant81995/introducing-gradient-descent-minimizing-the-cost-function-65fff3de8ac6

RAKHECHA, Aditya, 2019. Understanding Learning Rate. *Medium* [online]. [cit. 2024-03-28]. Dostupné z: https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de

ROY, Abhijit, 2020. An Introduction to Gradient Descent and Backpropagation. *Towards Data Science* [online]. [cit. 2024-03-27]. Dostupné z: https://towardsdatascience.com/an-introduction-to-gradient-descent-and-backpropagation-81648bdb19b2

RSVMUKESH, 2023. Determining the Number of Epochs. *Medium* [online]. [cit. 2024-03-28]. Dostupné z: https://medium.com/@rsvmukhesh/determining-the-number-of-epochs-d8b3526d8d06

SHARMA, Pranshu, 2023. Understanding Transfer Learning for Deep Learning. *Analytics Vidhya* [online]. [cit. 2024-03-27]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/

SHARMA, Sagar, 2017. What the hell is perceptron? In: *Medium* [online]. [cit. 2024-03-31]. Dostupné z: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

SHARMA, Siddharth, Simone SHARMA a Anidhya ATHAIYA, 2020. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*. **4**(12), 310-316. ISSN 2455-2143.

SHEN, Kevin, 2018. Effect of batch size on training dynamics. *Medium* [online]. [cit. 2024-03-28]. Dostupné z: https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e#:~:text=Finding%3A%20large%20batch%20size%20means,all%20about%20the%20same%20size

8 Seznam obrázků

9 Seznam grafů

10 Seznam příkladů kódu