Středoškolská technika 2022

Setkání a prezentace prací středoškolských studentů na ČVUT

Konvoluční neuronová síť pro klasifikaci dopravního značení

David Korčák

Gymnázium Josefa Jungmanna
Svojsíkova 1, 412 01 Litoměřice

Gymnázium Josefa Jungmanna, Litoměřice,
Svojsíkova 1, příspěvková organizace

Zadání maturitní práce ve školním roce 2021/2022

Téma maturitní práce:	Konvoluční neuronové sítě
Jméno, příjmení žáka/žákyně:	David Korčák
Třída:	8.O
Termín odevzdání:	31. 3. 2022
Vedoucí maturitní práce:	Pavel Beránek

Popis maturitní práce:
Cílem maturitní práce je vytvořit model konvoluční neuronové sítě schopný rozpoznat dopravní značky. K učení a vyhodnocení přesnosti modelu bude využita datová sada German Traffic Sign Recognition Benchmark. Volba jazyka a modulů je na žákovi, ale doporučené je využít jazyk Python3 a nějaký z široce využívaných modulů pro strojové učení (TensorFlow, PyTorch). Za funkční aplikaci (například řešení ve formě mobilní aplikace v chytrém telefonu), která bude schopna rozpoznávat dopravní značky v reálném provozu, jsou plusové body. Komisi bude poskytnuta demoverze pomocí vhodné simulace rozpoznávání dopravních značek. Pro vysvětlení bude komisi poskytnuta sada diagramů, dokumentujících architekturu a funkčnost aplikace.

Doporučené technologie:
1. Programovací jazyk Python3
2. Moduly pro tvorbu a učení modelů neuronových sítí (TensorFlow2, PyTorch)
3. Moduly pro datovou analýzu (Numpy, Pandas)
4. Moduly pro vizualizaci dat (Matplotlib, Seaborn)
5. Moduly pro zpracování obrazu (Open-CV, Scikit-learn, PIL)

Datum: 26. 11. 2021 **Podpis:**

Prohlášení

Prohlašuji, že jsem maturitní práci zpracoval samostatně a uvedl jsem všechny použité informační zdroje a literaturu v seznamu použitých zdrojů v souladu s normami a etickými principy závěrečných prací.

V Litoměřicích, 30. března 2022

Poděkování

Děkuji panu vedoucímu práce Mgr. Pavlovi Beránkovi za rady a podporu v průběhu její tvorby.

Abstrakt

Tato maturitní práce se zabývá vývojem, učením a aplikací konvoluční neuronové sítě pro klasifikaci dopravních značek. Pro učení sítě byla použita datová sada German Traffic Sign Recognition Benchmark, pro ověření jejího praktického potenciálu pak vlastnoručně získané snímky dopravních značek. Síť byla převedena do formátu kompatibilního s mobilním zařízením a demonstrována ve formě mobilní aplikace na určení dopravní značky z obrazové přenosu v reálném čase.

Klíčová slova: konvoluční neuronová síť, hluboké učení, GTSRB, dopravní značky, mobilní aplikace

Abstract

The objective of this graduation thesis was development, training, and application of a convolutional neural network for traffic sign classification. For neural network training, we used the German Traffic Sign Recognition Benchmark dataset. Real life performance was assessed using self-collected traffic sign images. Subsequently, the network was converted for mobile use and showcased as a part of a mobile application for traffic sign classification from real-time video feed.

Keywords: convolutional neural network, deep learning, GTSRB, traffic sign, mobile application

Title translation: Convolutional neural networks for traffic sign classification

Obsah

1. Úvod 12

2. Teorie 13

2.1 Strojové učení 13

2.1.1 Metody učení 13

2.1.2 Modely využívané ve strojovém učení 14

2.1.3 Některé úlohy a aplikace strojového učení 15

2.2 Umělé neuronové sítě 16

2.2.1 Výpočetní celky tvořící neuronové sítě 16

2.2.2 Učení neuronových sítí 20

2.2.3 Problémy učení neuronových sítí 22

2.2.4 Nástroje pro evaluaci výkonu neuronových sítí 23

2.2.5 Nástroje pro tvorbu neuronových sítí 24

2.2.6 Základní typy neuronových sítí 24

2.3 Konvoluční neuronové sítě 26

2.3.1 Architektura konvoluční neuronové sítě 26

2.3.2 Některé architektury konvolučních neuronových sítí 30

3. Realizace 31

3.1 Datová sada pro učení modelu 31

3.1.1 Segmentace a příprava dat pro učení neuronové sítě 32

3.2 Model konvoluční neuronové sítě 33

3.2.1 Volba architektury modelu 33

3.2.2 Úprava modelu 34

3.2.3 Učení modelu 36

3.3 Validace naučeného modelu 38

3.4 Praktická zkouška sítě na vlastních datech 39

3.5 Konverze modelu do formátu pro mobilní zařízení 40

3.6 Demonstrační mobilní aplikace 41

3.6.1 Volba cílové platformy 41

3.6.2 Demonstrační zařízení 41

3.6.3 Tvorba aplikace 41

4. Výsledky 43

4.1 Průběh učení neuronové sítě 43

4.2 Ověření výkonu neuronové sítě 47
4.3 Výsledky praktické zkoušky sítě na vlastních datech 48
4.4 Mobilní aplikace 49
4.4.1 Výpočetní výkon a energetická náročnost aplikace..... 49
4.4.2 Uživatelské rozhraní aplikace..... 49
4.4.3 Klasifikační výkon aplikace 52
5. Diskuze..... 56
6. Závěr 58
7. Reference 59
7.1 Informační zdroje 59
7.2 Obrazové zdroje..... 64
7.3 Softwarové zdroje 65

Zkratky

CNN – Convolutional Neural Network

GTSRB – German Traffic Sign Recognition Benchmark

TP – True Positive

FP – False Positive

TN – True Negative

FN – False Negative

RGB – Red Green Blue

MNIST - Modified National Institute of Standards and Technology

PPM – Portable Pixmap Format

JPEG - Joint Photographic Experts Group

API – Application Programming Interface

CSV – Comma-separated value

GPU – Graphics Processing Unit

CPU – Central Processing Unit

NPU – Neural Processing Unit

GB – Gigabyte

FPS – Frames per second

Obrázky

Obrázek 1: model umělého neuronu 16
Obrázek 2: graf průběhu funkce Sigmoida 17
Obrázek 3: graf průběhu funkce Softmax se dvěma vstupy, 3 dimenze 18
Obrázek 4: graf průběhu funkce ReLU 19
Obrázek 5: model perceptronu 24
Obrázek 6: graf průběhu a předpis krokové funkce 25
Obrázek 7: uspořádání vrstev vícevrstvého perceptronu 25
Obrázek 8: obecná architektura konvoluční neuronové sítě 26
Obrázek 9: konvoluce vstupu filtrem 27
Obrázek 10: prokládání vstupu 28
Obrázek 11: nulové prokládání vstupu 28
Obrázek 12: sdružovací vrstva s filtrem 2x2, porovnání funkce průměru a maxima 29
Obrázek 13: značky přítomné v datové sadě GTSRB 32
Obrázek 14: ukázkové snímky z datové sady 32
Obrázek 15:graf vývoje přesnosti sítě na učící složce – 1. fáze učení 43
Obrázek 16: graf vývoje hodnoty ztrátové funkce sítě na učící složce – 1. fáze učení 43
Obrázek 17: graf vývoje přesnosti sítě na validační složce – 1. fáze učení 44
Obrázek 18: graf vývoje hodnoty ztrátové funkce na validační složce – 1. fáze učení 44
Obrázek 19: graf vývoje přesnosti sítě na učící složce – 2. fáze učení 45
Obrázek 20: graf vývoje hodnoty ztrátové funkce sítě na učící složce – 2. fáze učení 45
Obrázek 21: graf vývoje přesnosti sítě na validační složce – 2. fáze učení 46
Obrázek 22: graf vývoje hodnoty ztrátové funkce na validační složce – 2. fáze učení 46
Obrázek 23: matice zmatení pro celou validační část sady GTSRB 47
Obrázek 24: fotografie značky Kruhový objezd 48
Obrázek 25: fotografie značky Zákaz vjezdu všech vozidel 48
Obrázek 26: fotografie značky Hlavní pozemní komunikace 48
Obrázek 27: nová ikona aplikace 49
Obrázek 28: úvodní obrazovka aplikace 50
Obrázek 29: hlavní obrazovka aplikace 51
Obrázek 30: klasifikace reálné značky v provozu 52
Obrázek 31: klasifikace reálné značky v provozu 53
Obrázek 32: klasifikace reálné značky v provozu 54
Obrázek 33: klasifikace reálné značky v provozu 55

Tabulky

Tabulka 1: architektura sítě, výstup modulu TensorFlow 35

Tabulka 2: porovnání akcelerátorů neuronových sítí na chipsetu A15 Bionic..... 49

Přílohy

Příloha 1: Tabulka číselného pořadí a názvů dopravních značek sady GTSRB 1

Příloha 2: Grafické znázornění architektury sítě z aplikace Netron 1

1. Úvod

Počítačové vidění je v dnešní době jedním z nejrychleji rostoucích oborů hlubokého učení. Schopnost zpracování obrazových dat strojem a jejich porozumění nalézá uplatnění v mnoha oborech, od místní správy až po autonomní vozidla. Počítačové vidění umožňuje automobilům obrazová data z kamer a jiných vizuálních senzorů pochopit a činit rozhodnutí, jak se v provozu pohybovat. Systém musí být schopný orientace v prostředí pozemních komunikací tak, jak bylo původně konstruováno pro lidské řidiče. Jedním z hlavních prvků, jenž nastavuje pravidla provozu, jsou dopravní značky. Automobil je musí bezchybně rozeznat a řídit se jimi stejně, jako lidští řidiči.

V dnešní době je ve strojovém učení k počítačovému vidění stále více využíváno konvolučních neuronových sítí. Jedná se v této oblasti o nejúspěšnější a nejefektivnější matematické modely. I přes překážky týkající se jejich učení se jedná o nejvhodnější způsob, jakým řešit úlohu rozpoznávání dopravního značení.

Cílem této práce je vývoj a demonstrace konvoluční neuronové sítě, která bude schopná z obrazových dat klasifikovat dopravní značky do jedné z 43 kategorií. Vedle úspěšnosti sítě na vlastních datech budeme také zkoumat kvantifikaci přesnosti podle statistických funkcí a přesnost vůči ostatním sítím používaných pro stejnou úlohu. Demonstračním nosičem bude aplikace pro mobilní telefon, jehož fotoaparát bude simulovat kameru automatizovaného vozidla. Díky pokrokům v mikročipových technologiích můžeme využít dedikovaný hardware, který moderní mobilní zařízení nabízejí pro zlepšení výkonu modelů strojového učení. To zároveň simuluje optimální výpočetní výkon, který nabízí technika autonomního vozidla. U aplikace bude předmětem pozorování její výkon, energetická náročnost a praktická užitnost. Samotná síť byla implementována v programovacím jazyce Python pomocí modulu TensorFlow, resp. TensorFlow Lite pro mobilní zařízení.

2. Teorie

2.1 Strojové učení

Strojové učení je oblast počítačových algoritmů, které se ve své určené aplikaci zlepšují a zefektivňují svoji činnost podle daných kritérii. Těmi jsou například přesnost určení výsledku, jeho rychlejší vyhodnocení nebo kombinace obou ve specifikovaném poměru. Strojové učení se řadí do oboru umělé inteligence, jeho konkrétní aplikace, například neuronové sítě, slouží jako prostředky k tvorbě umělé inteligence. Hlavní rozdíl mezi explicitně tvořenými algoritmy a těmi spadajícími do strojového učení je, že algoritmy strojového učení jsou pouze instrukcemi pro počítač (stroj) ke hledání vlastního algoritmu. Není se tak již nutné zabývat tvorbou každé z velkého množství instrukcí potřebných pro komplexní aplikace jako rozpoznávání obrazu, zvuku či analýzu a předpověď na základě trendů v datech. Samotný proces, kterým počítač hledá optimální algoritmus pro splnění dané úlohy, se nazývá učení [1][2].

2.1.1 Metody učení

Učení s učitelem

Metoda učení s učitelem využívá označenou množinu dat, pomocí které je algoritmu specifikováno, jaké proměnné má ze vstupu určit a jaký je správný výstup. Po vyhodnocení výsledku algoritmus spočítá, jak daleko byl od cílené výstupní hodnoty a na základě toho, pomocí optimalizačního algoritmu, upraví svoji operaci, aby se chtěnému výsledku co nejvíce přiblížil. Příklad tohoto principu učení je neuronová síť se sadou dat ručně psaných číslic MNIST [4]. Každý z 60 000 obrázků má atribut správného výstupu, tzv. „label“. Algoritmus během učení využívá tento referenční výstup pro porovnání s tím svým [1][2].

Učení bez učitele

Tento typ učení nelze využívat k přesné klasifikaci dat, jako například určení psané číslice či druhu zvířete ve fotografii. V tomto případě algoritmus v datech pouze hledá podobnost a opakující se trendy. Na jejich základě data rozdělí do skupin s podobnými rysy a data, která mají být určena, zařadí do jedné z oněch skupin či předpoví hodnotu pro nějakou vstupní proměnnou. Mezi metody využívající učení bez dohledu patří například regresní modely [1][2].

Posilované učení

Optimalizace posilovaným učení funguje na principu odměny, ať již ve formě bodů, hodnoty funkce či obdobné kvantifikovatelné veličiny. Programy a algoritmy se svým chováním snaží maximalizovat zisk či hodnotu odměny a pro tento účel své chování měnit. Pomocí posilovaného učení lze modelovat například sociologický model teorie her či naučit algoritmus ovládat autonomní vozidlo. V obou případech je učitelem dodána odměna ve formě správného rozhodnutí, čímž algoritmus maximalizuje jejich provedený počet [1][2].

2.1.2 Modely využívané ve strojovém učení

Neuronové sítě

Matematický model tvořený propojením velkého množství výpočetních jednotek, neuronů. Neurony si navzájem předávají hodnoty svých výstupů a jsou uspořádány do vrstev, vstupní, skrytých a výstupní. Neuron jako takový představuje lineární kombinaci všech svých vstupů, jejíž hodnota je tzv. aktivační funkcí převedena na samotný výstup neuronu, který slouží jako vstup pro neurony v další vrstvě. Propojení neuronů jsou upravována váhami, hodnotami „důležitosti“ daného spoje pro neuron, jemuž spoj slouží jako vstup. Koncept neuronových sítí je inspirován propojením nervových buněk ve organickém mozku. V dnešní době se jedná o nejhojněji využívané modely strojového učení, používají se ke klasifikaci (analýza obrazu, zvuku), regresi (ceny nemovitostí) i v robotice v kombinaci s posilovaným učením (autonomní vozidla, letadla) [5].

Regresní analýza

Široké spektrum metod využívaných k nalezení modelu, který co nejpřesněji předpoví znaky či hodnoty na základě vstupních proměnných. Nejjednodušším příkladem je lineární regrese, která proloží množinu bodů přímkou, lineární funkcí, jenž pro nové vstupní proměnné co nejpřesněji určí výsledek tak, aby odpovídal trendům výchozí množiny bodů. K nalezení rovnice přímky se využívají matematická kritéria ve formě chyb, vzdáleností všech bodů od dané přímky. S lineární regresí je nejčastěji využívána metoda nejmenších čtverců, jejíž podstatou je minimalizace součtu všech druhých mocnin vzdáleností jednotlivých bodů od přímky. Regresní analýza a metody strojového učení jí využívající nalézají uplatnění primárně na finančních trzích (předpověď vývoje cen, trhů) a trhu s nemovitostmi (předpověď ceny nemovitosti na základě rozlohy, polohy) [3][8][9].

Ostatní modely

Mezi ostatní modely využívané ve strojovém učení se řadí například rozhodovací stromy [6][26] a podpůrné vektory [7][27].

2.1.3 Některé úlohy a aplikace strojového učení

Klasifikační úlohy a jejich aplikace

Klasifikace je úloha statistiky, kdy dochází k zařazení vzorku (datového bodu) do dané třídy (kategorie) na základě pozorování jiných vzorků s již známým zařazením. Rozřazování je prováděno na základě vstupních proměnných (vlastností dané instance, kterou je potřeba zařadit). Klasifikace je prováděna výhradně pomocí učení s učitelem, kdy je nutné naučit model rozeznávat jednotlivé konkrétní kategorie. Stejné algoritmy lze však použít s učením bez učitele. V takovém případě se jedná o tzv. klastrování („clustering“, seskupování). Klasifikace nachází své uplatnění ve velkém množství různých odvětví. Za zmínku stojí například detekce znaků v obrazu, nejčastěji číslic, písmen a objektů (zvířat, lidí, automobilů). S nejjednodušší klasifikací se lze setkat například i u poskytovatelů emailových schránek, kde jsou zprávy na základě obsahu řazeny mezi důležité, marketingové nebo spamové [3].

Klastrování (shlukování) a jeho aplikace

Shlukování je odvozeno od metody klasifikace, zásadní rozdíl je však v kategoriích, do kterých jsou data řazena. Zatímco klasifikace rozřazuje data do konkrétních skupin na základě konkrétních znaků, shlukování pouze seskupuje data s podobnými znaky. Podmínkou pro úspěšnou aplikaci klastrování je, aby byl každý z množiny sledovaných bodů charakterizován stejnými znaky (např. výška a váha osoby) a zároveň společným znakem, který lze pozorovat u většího množství bodů z dané množiny (např. vyšší lidé s větší hmotností, menší lidé s nižší hmotností). Výsledkem analýzy je pak několik skupin, na které se zkoumaná data dělí. Jedná se o vhodný nástroj pro marketing a obchodní strategie, kdy je možné na základě chování zákazníků odvodit zhruba jaké typy spotřebitelů v našem podniku nakupují. Pak lze cíleně inzerovat těmto skupinám či pro ně vytvářet zboží, optimálně podle jejich chování tak, abychom dosáhli co nejvyššího zisku [10].

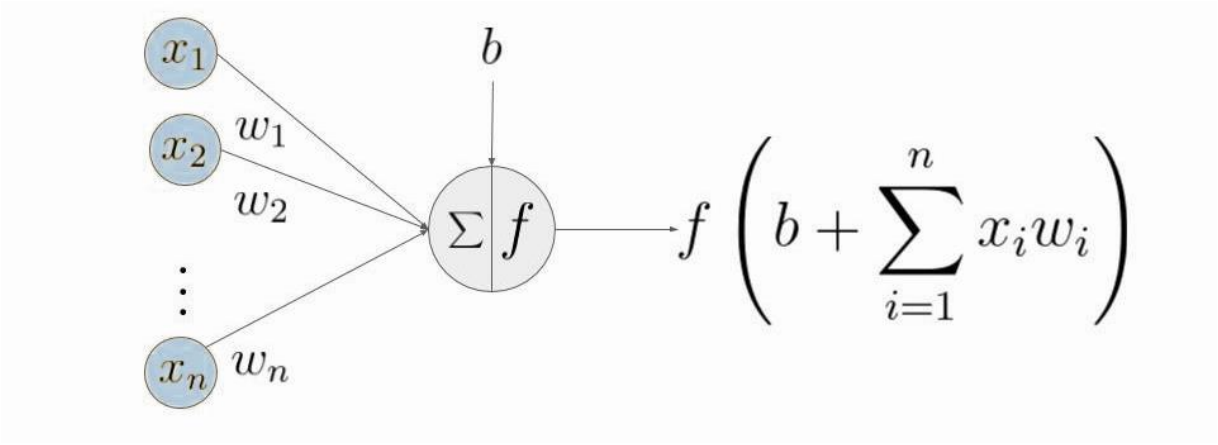
2.2 Umělé neuronové sítě

Umělá neuronová síť (dále jen neuronová síť) je výpočetním matematickým modelem, jehož podstatou je velké množství plně propojených výpočetních jednotek, neuronů. Ty jsou uspořádány do vrstev, jednotlivé neurony jsou pak mezi sebou propojeny. Zpravidla má umělá neuronová síť vstupní vrstvu, jednu nebo více skrytých vrstev a vrstvu výstupní. Všechny hodnoty, se kterými neuron a neuronová síť pracují, jsou reálná čísla. Neuronová síť sama o sobě je pouze vysokoúrovňovým algoritmem, jenž se pro danou konkrétní aplikaci učí sám. Učení může být prováděno metodami, které byly vymezeny v oddílu 1.1. Pro učení je klíčový koncept zpětné propagace. Jedná se o základní matematický mechanismus, díky němuž je možno v neuronové síti během učení měnit váhy, parametry, jejichž hodnoty budou pro každou úlohu unikátní [5][11][12].

2.2.1 Výpočetní celky tvořící neuronové sítě

Neuron

Neuron je základní výpočetní prvek neuronové sítě. Zpravidla se skládá ze dvou částí. Vstupní část zahrnuje hodnoty všech jeho jednotlivých vstupů, upravených tzv. váhami. Váhy jsou reálná čísla určující, jak velkou roli bude hrát hodnota konkrétního vstupu na jejich celkovém součtu. Součet všech vstupních hodnot upravených váhami je prostá lineární kombinace. Z důvodu vyšší přesnosti se k součtu přičítá tzv. bias, konstantní reálné číslo upravující konečnou hodnotu lineární kombinace. Ta je následně využita v druhé části neuronu. Druhá část neuronu je tvořena aktivační funkcí. Aktivační funkce přebírá jako vstup lineární kombinaci vstupů z první části neuronu. Její hodnota je pak výstupní hodnotou celého neuronu a je předána neuronům v další vrstvě [5][12].



Obrázek 1: model umělého neuronu [Obr. 1]

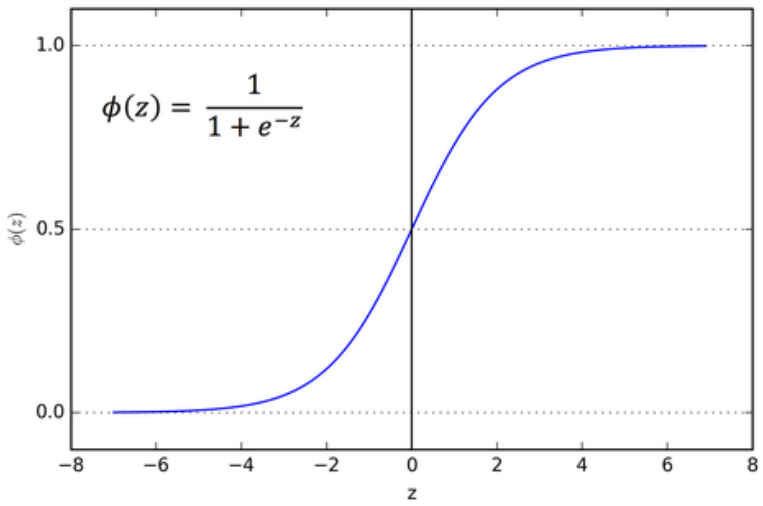
Aktivační funkce neuronu

Účelem aktivační funkce je převést vstup neuronu na jedinou výstupní hodnotu. Vzhledem k tomu, že se neuronové sítě využívají v mnoha aplikacích s odlišnými požadavky, je nutné zohlednit a optimalizovat jejich části pro tyto případy. Aktivační funkce je jedním z hlavních prostředků, jimiž lze optimalizace dosáhnout. Dělí se na dva typy, lineární a nelineární. Lineární aktivační funkci je možno také nazvat funkcí identity, jelikož její výstup se rovná vstupu. Je to nejprostší funkce, kterou lze jako aktivační použít, avšak v moderních neuronových sítích se vyskytuje velmi zřídka. Nelineárních aktivačních funkcí existuje celá řada a nalezneme je ve většině moderních neuronových sítích [12][13].

Nejčastější typy aktivačních funkcí

Sigmoida

Sigmoida [28], funkce, jejíž graf vypadá jako písmeno s, je často využívaná v neuronových sítích s pravděpodobnostním výstupem. Obor hodnot této funkce je $(0; 1)$, což odpovídá intervalu možné pravděpodobnosti jevu.



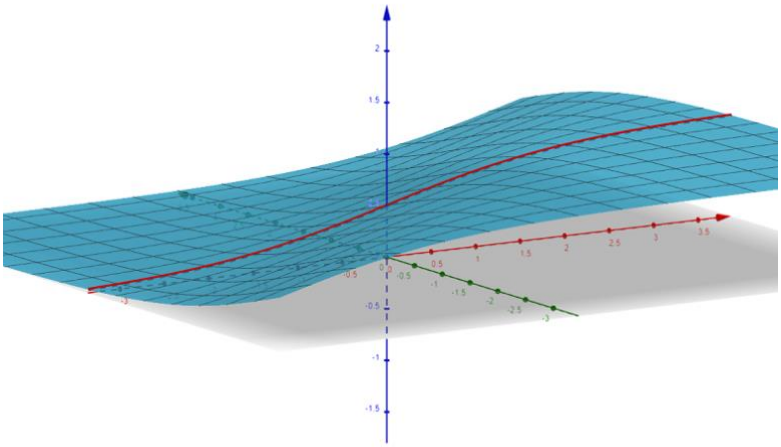
Obrázek 2: graf průběhu funkce Sigmoida [Obr. 2]

$$S(x) = \frac{1}{1 + e^{-x}}$$

Předpis funkce Sigmoida

Softmax

Funkce Softmax [28], někdy také normalizovaná exponenciální funkce, je podobná sigmoidě až na rozdíl, že pro n vstupních hodnot vrací n výstupních pravděpodobností. Výsledek použití funkce je tedy podobný, jako vypočítat n hodnot n funkcí sigmoida pro n vstupních proměnných. Její výstup je opět pravděpodobnostní a jedná se o funkci preferovanou pro úlohy klasifikace více tříd (více výstupních hodnot funkce).



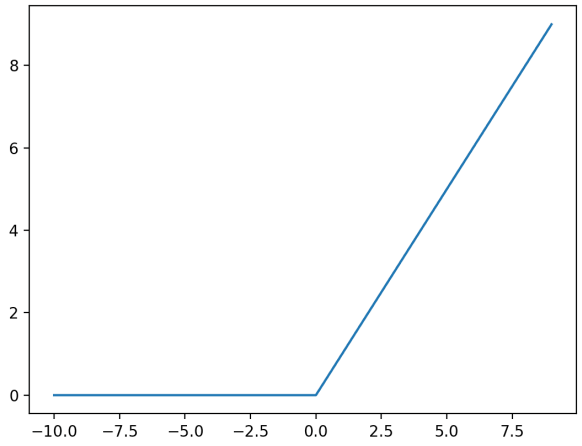
Obrázek 3: graf průběhu funkce Softmax se dvěma vstupy, 3 dimenze [Obr. 3]

$$\sigma(\vec{x}_i) = \frac{e^{x_i}}{\sum_{j=1}^{M} e^{x_j}}$$

Předpis funkce Softmax

ReLU

Rektifikovaná lineární jednotka je speciální funkcí, která se pro kladné vstupy chová jako prostá funkce identity $f(x) = x$, avšak pro vstupní proměnné nižší nebo rovny nule je její hodnota nulová. Její obor hodnot je oproti prosté funkci identity omezený na $< 0; \infty)$. Dnes se jedná o nejběžněji využívanou aktivační funkci.



Obrázek 4: graf průběhu funkce ReLU [Obr. 4]

$$f(x) = \max(0, x)$$

Předpis funkce ReLU

Vstupní vrstva

Informace na vstupu neuronové sítě jsou zpracovány první, vstupní, vrstvou. Vstup neuronů v této vrstvě je pouze jedna hodnota, např. intenzita jasu jednoho obrazového bodu. Jejich výstup je pak předán každému z neuronů v následující vrstvě.

Skryté vrstvy

Mezi vstupní a výstupní vrstvou neuronové sítě se nachází jedna nebo více skrytých vrstev. Neurony v nich obsažené jsou mezi sebou plně propojené, každý z nich má vstup ze všech předchozích neuronů a jeho výstup náleží každému z neuronů ve vrstvě následující.

Výstupní vrstva

Poslední vrstva obsahuje neuronů stejně, jako existuje výstupních proměnných (tříd, obrazových bodů atp.). Aktivační funkce výstupní vrstvy se volí tak, aby co nejlépe vystihla vztah proměnných ke vstupním datům a ta mohla být co nejlépe klasifikována. Číselné pořadí tříd je shodné s číselným pořadím neuronů, např. pro síť rozpoznávající číslice z databáze MNIST [4] by měla výstupní vrstva 10 neuronů (číslice 0-9). Nejvyšší aktivace na 3. neuronu by pak znamenala, že vstup byl klasifikován jako číslice 2.

2.2.2 Učení neuronových sítí

Mechanismus zpětné propagace

Zpětná propagace je základním matematickým algoritmem, pomocí kterého lze upravovat hodnoty parametrů neuronových sítí vzhledem k jejich výstupu a odchylky výstupu od cílené hodnoty. Zpětná propagace je založena na 3 základních pojmech, ztrátové funkci (error function), propagaci chyby a optimalizačním algoritmu [14].

Ztrátová funkce

Za účelem kvantifikace chyb, které neuronová síť dělá při výpočtu a následně se projevují na finálním výstupu se používá ztrátová funkce. Vstupními proměnnými ztrátové funkce jsou aktuální výstup neuronové sítě a cílený výstup (učení s učitelem). Ztrátová funkce musí být diferencovatelná kvůli své klíčové roli v mechanismu zpětné propagace. Jednou z nejpopulárnějších ztrátových funkcí je například průměr druhých mocnin odchylek (MSE – mean squared error):

$$\frac{1}{N}\sum_{i=1}^{N}(\widehat{y_i} - y_i)^2$$

S ní související je i funkce průměru absolutních hodnot odchylek (MAE – mean absolute error):

$$\frac{1}{N}\sum_{i=1}^{N}|\widehat{y_i} - y_i|$$

Funkcí často využívanou pro modely klasifikující více tříd, jejichž výstup je vektor pravděpodobností od 0 do 1 je Cross-entropy loss (logaritmická ztrátová funkce):

$$-\sum_{c=1}^{M} y_{o,c} \times \ln(p_{o,c})$$

kde M počet výstupních tříd sítě, y je binární hodnota určující, zda měla být třída c přítomna na výstupu (1 pokud ano, 0 pokud ne) při pozorování o a p pravděpodobnost přítomnosti dané třídy c na výstupu (výstup sítě s funkcí Softmax). Protože pro každou třídu, jenž na výstupu přítomna být neměla, bude hodnota y rovna nule, lze předpis funkce výrazně zjednodušit.

$$-\ln(p_{o,c})$$

Pro samotné učení sítě je výhodou logaritmický průběh funkce [40]. Její hodnota bude při špatné klasifikací velmi vysoká, což se projeví i na hodnotě její derivace. Umožní to tak během zpětné propagace rychle optimalizovat váhy, jakmile ale začne chyba klesat, hodnota funkce (a její derivace) začnou klesat pomaleji a umožní jemné úpravy vah tak, aby bylo nalezeno globální minimum funkce (nejnižší možná hodnota chyby).

Propagace chyby

Po výpočtu chyby výstupu sítě dochází k výpočtům pomocí algoritmu zpětné propagace. Ten hledá, jak jednotlivé parametry sítě ovlivňují konečnou hodnotu ztrátové funkce. Chybu sítě budeme parciálně derivovat podle každého z parametrů a následně určovat numerické hodnoty těchto derivací. Hodnoty záleží vždy na vrstvě předchozí, je tedy nutné začínat od vrstvy poslední a postupovat až k té vstupní.

Optimalizační algoritmy

Po výpočtu chyby sítě a všech derivací podle parametrů, které chybu ovlivňují, je na řadě upravit jednotlivé parametry tak, aby byla chyba na výstupu minimalizována. Nejhojněji využívanými algoritmy jsou gradienty (Gradient Descent). Výpočtem derivací chyby podle parametrů dostaneme jednotlivé prvky vektoru gradientu. Od vektoru aktuálních parametrů pak odečteme vektor gradientu upravený velikostí učícího kroku. Pokud by byl učící krok moc veliký či by se v algoritmu nevyskytoval, hrozilo by, že nikdy nedojde k nalezení pravého minima funkce. Pro každý parametr v neuronové síti tedy platí:

$$\Delta w_{ij}^{k} = -\alpha \frac{\delta E(X,\theta)}{\delta w_{ij}^{k}}$$

kde w je aktuální parametr, α je rychlost učení a E je hodnota ztrátové funkce. Zápis lze zjednodušit na celé vektory:

$$\boldsymbol{x}(p)_{i+1} = \boldsymbol{x}(p)_{i} - \alpha \nabla E(X,\theta)$$

kde x(p) je vektor všech parametrů sítě. Standardní gradientní algoritmus existuje i ve verzi se setrvačností (Momentum). Ta do rovnice přidává faktor setrvačnosti závislý na velikosti předchozích změn vah:

$$\boldsymbol{x}(p)_{i+1} = \boldsymbol{x}(p)_{i} - \alpha \nabla E(X,\theta) + \beta(\boldsymbol{x}(p)_{i} - \boldsymbol{x}(p)_{i-1})$$

kde β je hyper-parametr setrvačnosti. Hodnota setrvačnosti pomůže algoritmu překonat lokální minima a nalézt pravé globální minimum.

Jedním z nejnovější a nejúčinnějších algoritmů pro optimalizaci parametrů sítě je Adam (Adaptive Moment Estimation) [41][42]. Tento algoritmus mimo hodnot popsaných v předchozích odstavcích sleduje gradienty samotných parametrů a zachovává optimalizační hyper-parametry pro každý parametr sítě individuálně.

2.2.3 Problémy učení neuronových sítí

Během učení může dojít k řadě nežádoucích jevů, které způsobí, že síť nedosáhne tak vysoké přesnosti. Mohou být způsobeny nedostatečnou optimalizací, nekvalitními vstupními daty či naopak přílišným učením a optimalizací na trénovací sadě.

Lokální minimum ztrátové funkce

Pokud dojde k nalezení lokálního minima ztrátové funkce a algoritmu se ho nepodaří překonat, přesnost sítě se již nebude dalším učením zvyšovat. Jedná se o jev spojený s mnohadimenzionální povahou vektorových prostorů, ve kterých k hledání minima funkce dochází. Moderní algoritmy, jako například zmiňovaný Adam, jsou však navrženy s lokálními minimy a dokážou je překonat

Bias (předpojatost)

Předpojatost modelu může vzniknout, pokud se nachází již ve vstupních datech. V případě, že jedna třída bude zastoupena výrazně více než jiná, výsledný model pak bude hůře klasifikovat méně zastoupenou třídu a častěji chybně umisťovat datové body do třídy, jež byla při učení početnější.

Přeučení (overfitting)

V některých případech učení může dojít k tzv. přeučení sítě. Při tomto jevu jsou parametry sítě „přeoptimalizovány“ na trénovací sadu dat. Síť pak úspěšně klasifikuje data v ní obsažená, ale má nízkou přesnost na prvcích mimo trénovací množinu.

2.2.4 Nástroje pro evaluaci výkonu neuronových sítí

Za účelem statistického popisu úspěšnosti modelů neuronových sítí existuje mnoho metrik. Ve většině případů se metriky odvíjejí od četností správně identifikovaných dat (True Positive, True Negative), kladně identifikovaných negativních dat (False Positive) a negativně identifikovaných kladných dat (False Negative). Statistické funkce pak využívají četnosti těchto jevů k popisu přesnosti modelů [48].

Přesnost (accuracy)

Hodnota přesnosti reprezentuje poměr správně provedených klasifikací model (True Positive, True Negative) a počtu všech přítomných vzorků.

$$A = \frac{počet\ správných\ klasifikací}{počet\ všech\ datových\ vzorků}$$

Precision

Metrika precision nám říká, jak často síť provede správnou (True Positive) kladnou klasifikaci dané třídy v porovnání se všemi kladnými klasifikacemi pro danou třídu (True Positive, False Positive).

$$P = \frac{počet\ správných\ kladných\ klasifikací\ (TP)}{počet\ všech\ kladných\ klasifikací\ (TP + FP)}$$

Recall

Recall reprezentuje, kolikrát byla ze všech vzorků dané třídy (True Positive, False Negative) kladně klasifikována jako daná třída (True Positive).

$$R = \frac{počet\ správných\ kladných\ klasifikací\ (TP)}{počet\ všech\ vzorků\ dané\ třídy\ (TP + FN)}$$

Matice zmatení

Matice zmatení je vizuální pomůckou pro posouzení přesnosti sítě. Jedná se o maticový graf, kde jedna z os představuje správné známé třídy a druhá výstup modelu. Jednotlivá políčka jsou průsečíky os daných tříd, tedy jejich klasifikace. Průsečná políčka stejných tříd tvoří diagonálu matice. Následně se zpravidla barvou, někdy i číselně, v políčkách indikuje četnost výskytu této klasifikační kombinace (např. třída 2 byla klasifikována jako třída 2 nebo třída 5

byla klasifikována jako třída 11). Čím intenzivnější zabarvení pole, tím častěji se kombinace na výstupu vyskytuje. Ideální model by měl všechny hodnoty umístit na diagonálu matice, tedy jeho výstup by plně odpovídal známému správnému výstupu.

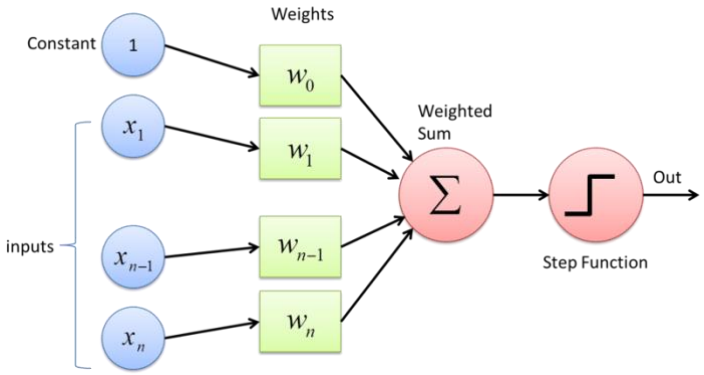
2.2.5 Nástroje pro tvorbu neuronových sítí

K tvorbě modelů neuronových sítí se v praxi využívá velkého množství knihoven a modulů pro různé programovací jazyky. Pro datovou analýzu a strojové učení se nejčastěji využívá jazyk Python. Existuje řada frameworků pro tvorbu modelů v jazyce Python, nejpopulárnějšími jsou TensorFlow, PyTorch a Keras. Předností knihovny TensorFlow je podpora velkého množství programovacích jazyků (C#, C++, Java, JavaScript, Swift) a mobilních platforem (Android, iOS).

2.2.6 Základní typy neuronových sítí

Jednovrstvý perceptron

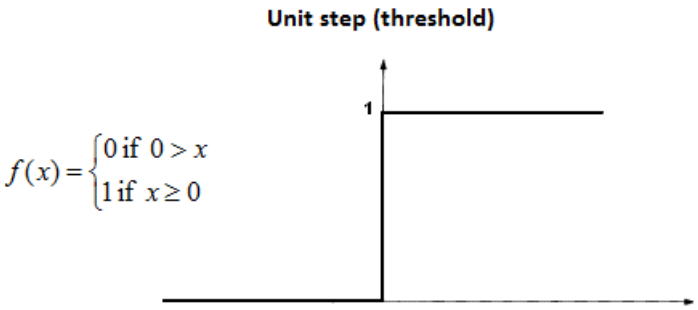
Nejjednodušší typ algoritmu, jenž je předchůdcem moderních neuronových sítí. Tvoří ho jeden umělý neuron s krokovou aktivační funkcí. Jedná se o prostý binární klasifikátor, který na základě vstupních dat určí, zda daný datový bod patří do jedné ze dvou tříd. Učení perceptronu musí probíhat s učitelem a je značně limitován typem dat. Ta musí být lineárně oddělitelná (tedy obsahovat přesně vymezené hranice mezi třídami), jinak nikdy nedojde k přesnému naučení perceptronu a úspěšné klasifikaci dat.



Obrázek 5: model perceptronu [Obr. 5]

Perceptron se skládá dvou částí. Na vstupu perceptronu jsou váhy pro jednotlivé proměnné a konstantní hodnota biasu. Výstupní část je kroková funkce, která určuje rozdělení datových bodů do jedné ze dvou tříd [15].

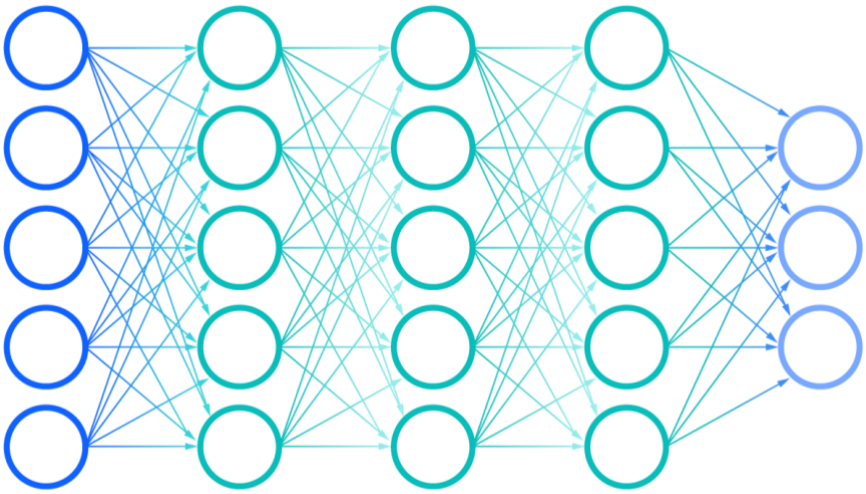
$$x = \sum_{i=1}^{N} (w_i x_i) + b$$



Obrázek 6: graf průběhu a předpis krokové funkce [Obr. 6]

Vícevrstvý perceptron

Vícevrstvý perceptron je základní neuronová síť obsahující vstupní, výstupní a jednu nebo více skrytých vnitřních vrstev. Dnes je také známý jako umělá neuronová síť. Lineární kombinace vstupů zpracovaná aktivační funkcí je použita jako vstup pro další skrytou vrstvu. Poslední vrstva pak dodává konečný výstup sítě.

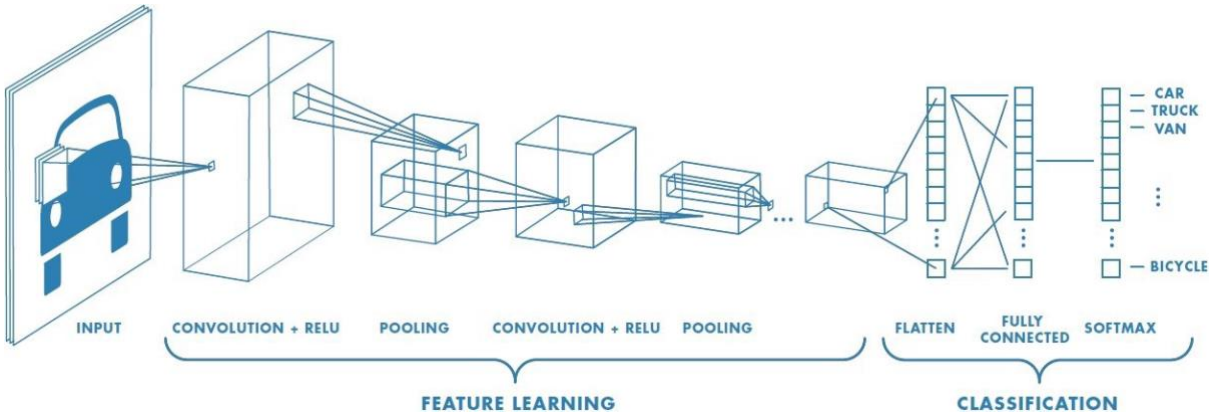


Obrázek 7: uspořádání vrstev vícevrstvého perceptronu [Obr. 7]

Je složen z velkého množství neuronů podobných perceptronu. Rozdíl je ten, že aktivační funkce použité v neuronech musejí být derivovatelné. Příčina tohoto požadavku je, že učení vícevrstvého perceptronu probíhá pomocí zpětné propagace [12][15].

2.3 Konvoluční neuronové sítě

Konvoluční neuronová síť je zdokonalením tradičních vícevrstvých perceptronů. Její koncepce vychází z lidského zrakového centra, hlavní oblastí uplatnění je rozpoznávání a klasifikace obrazových dat. Na rozdíl od vícevrstvého perceptronu obsahuje konvoluční neuronová síť konvoluční a sdružovací vrstvy. Konvoluční vrstva, první komponenta konvoluční neuronové sítě, je tvořena tzv. filtrem (jádrem), jenž ve vstupních datech hledá naučené znaky. Výsledkem této operace je mapa znaků, matice hodnot reprezentujících výskyt znaků v dané části obrazu. Sdružovací vrstva sítě potom pomocí matematických funkcí redukuje rozměr dat tak, aby bylo méně výpočetně náročné aplikovat další kroky. Cyklus těchto operací se provádí několikrát aby byla síť v dané aplikaci schopna rozlišit co nejvíce podrobných znaků s dostatečnou přesností. Klasifikační část obsahuje plně propojené vrstvy podobné jako ve vícevrstvých perceptronech.



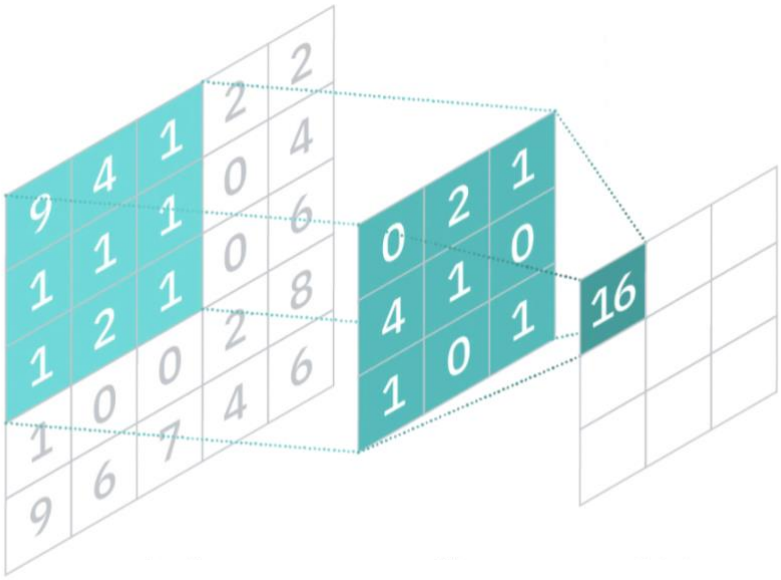
Obrázek 8: obecná architektura konvoluční neuronové sítě [Obr. 8]

2.3.1 Architektura konvoluční neuronové sítě

Konvoluční vrstva

Konvoluční vrstva sítě je základním elementem, který umožňuje síti prostorově nezávislé rozpoznávání znaků. Konvoluce je v kontextu sítě definována jako maticová operace. Jedna z matic, konvoluční filtr, je typicky čtvercová, a v porovnání se vstupním obrazem má výrazně menší rozměry. Nejčastěji se používá rozměr 3×3 nebo 5×5. Hodnoty matice filtru reprezentují váhy, reálná čísla udávající důležitost, a pořadí vah v matici pak samotné znaky. Matice filtru je iterativně posouvána po matici vstupních dat o vzdálenost kroku (stride). Nízké hodnoty kroku umožňují maximálnímu zachycení znaků v datech, vyšší hodnoty pak

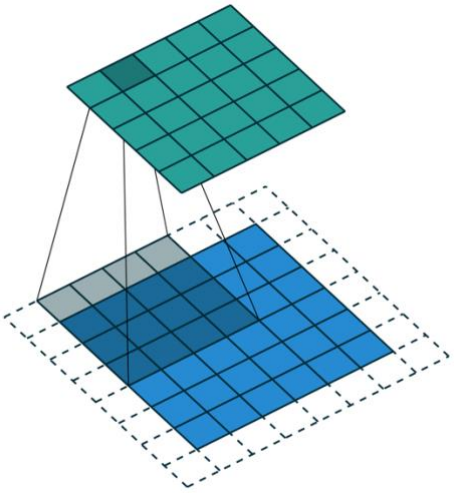
dimenzionálně menší výstup. Samotná operace konvoluce je prováděna jako součin hodnot matic na stejných pozicích. V případě velké matice vstupních dat se jedná o polohy v překryvu vstupu s konvolučním filtrem [16][17][18].



Obrázek 9: konvoluce vstupu filtrem [Obr. 9]

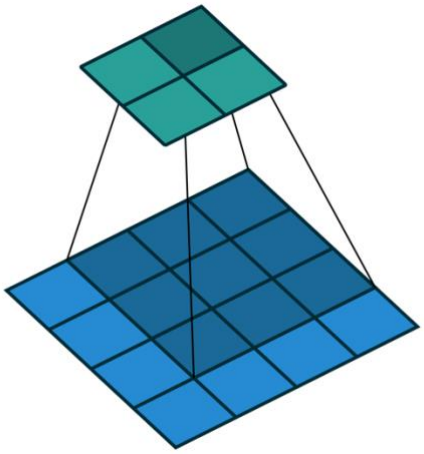
$$y_{i,j} = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} f_{k+1,l+1} x_{i+k,j+l}$$

Kdy $y_{i,j}$ je i,j-tý prvek výstupní matice, n velikost konvolučního filtru, f hodnota daného prvku filtru a x daná hodnota vstupní matice. Počet filtrů v konvoluční vrstvě závisí na hloubce výstupu. Pro obrazová data s třemi barevnými kanály (RGB) by konvoluční vrstva obsahovala 3 filtry, každý pro jeden barevný kanál. Posledním důležitým aspektem konvoluční vrstvy je prokládání (padding). Jedná se o techniku, která ovlivňuje velikost výstupní matice. Stejné prokládání (same padding) znamená, že vstupní matice je rozšířena tak, aby po konvoluci filtrem byla velikost výstupu stejná jako velikost původního vstupu.



Obrázek 10: prokládání vstupu [Obr. 10]

V případě nulového prokládání (valid padding) je filtr aplikován pouze na vstupní matici bez jakéhokoli rozšiřování. Poslední typ plného prokládání (full padding) rozšíří vstupní matici o tolik, aby výstupní matice měla větší rozměry než původní vstup.



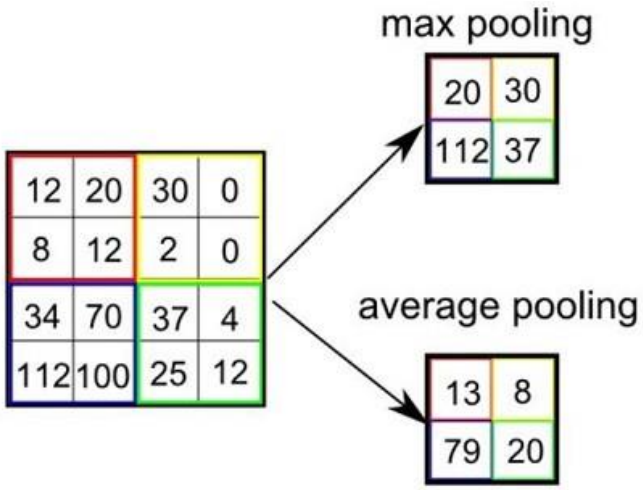
Obrázek 11: nulové prokládání vstupu [Obr. 11]

Váhy filtru jsou podobně jako parametry vícevrstvého perceptronu v průběhu učení upravovány zpětnou propagací, čímž dochází k samovolnému naučení rozpoznávání znaků v datech

.

Sdružovací vrstva

Sdružovací vrstva má v konvoluční neuronové síti 2 hlavní úlohy. Jednou je redukce velikosti matice s daty a druhou je zvýraznění znaků, které síť v datech hledá. Zmenšení dat představuje výhodu ve snížené výpočetní náročnosti, což může v reálné aplikaci ušetřit čas a energii. Zvýraznění znaků je jedním ze způsobů, jak zvýšit efektivitu jejich rozeznávání. Zároveň není v tomto procesu detekce znaků ovlivněna orientací v prostoru, takže síť dokáže vytáhnout i převrácené nebo rotované znaky. V této vrstvě je využíváno dvou hlavních technik, sdružování podle maxima a podle průměru. Podobně jako u konvoluční vrstvy, i sdružovací vrstva je aplikována formou filtru o daných rozměrech. Na hodnoty v překryvu filtru a vstupní matice je použita daná funkce a výstup předán do další vrstvy sítě.



Obrázek 12: sdružovací vrstva s filtrem 2x2, porovnání funkce průměru a maxima [Obr. 12]

Plně propojená vrstva

Plně propojená vrstva konvoluční neuronové sítě je běžnou neuronovou sítí, vícevrstvým perceptronem, která využívá jako vstup data upravená předchozími vrstvami. Techniky pro učení a tvorbu plně propojené vrstvy jsou shodné s vícevrstvým perceptronem definovaném v oddílu 2.2.2.

2.3.2 Některé architektury konvolučních neuronových sítí

LeNet-5 CNN

Síť typu LeNet byla jednou z prvních prakticky aplikovatelných konvolučních neuronových sítí. Finální iterace z roku 1998 označena číslem 5 je dnes jednou ze základních architektur CNN pro detekci ručně psaných číslic. Síť tvoří 3 konvoluční vrstvy, 2 sdružovací vrstvy a 2 plně propojené vrstvy. Vstupní vrstva má rozměry $32 \times 32 \times 1$ bodů, například pro datovou sadu MNIST [4] (obrázky $28 \times 28 \times 1$) dochází k prokládání nulovými body po okrajích. První vrstva obsahuje 6 konvolučních filtrů o rozměrech 5×5, výstupem je tedy 6 matic, map znaků (feature maps), o rozměrech 28×28. Druhá sdružovací vrstva pak redukuje data na 6 matic o rozměrech 14×14 prostřednictvím filtru průměru o rozměrech 2×2. Třetí konvoluční vrstva pak filtry o rozměrech 5×5 výstup převede na 16 matic 10×10 bodů. Čtvrtá sdružovací vrstva redukuje data filtrem 2×2 na matice o jednotlivých rozměrech 5×5. Finální konvoluční vrstva pak filtry 5×5 redukuje data na 120 číselných hodnot (matice 1×1). Plně propojené vrstvy klasifikují výstup do tříd. Oproti klasifikátorům datové sady MNIST bez konvolučních komponent (běžné vícevrstvé perceptrony) je tato síť výpočetně účinnější a přesnější [20][25].

Sítě typu MobileNet

Sítě typu MobileNet byly vyvinuty specificky pro použití v mobilních zařízeních. Ty představují výzvu z hlediska výpočetní náročnosti algoritmů, jelikož pracují s omezeným výkonem a množstvím energie. Pro redukci výpočetní náročnosti a snížení požadavků na operační paměť využívá MobileNet hloubkově dělitelnou konvoluci [22][23]. Hloubkově dělitelná konvoluce se provádí ve dvou krocích. První z nich, samotná hloubková konvoluce, je prováděna filtrem $D_k \times D_k \times 1$ na vstupních datech o rozměrech $D_i \times D_i \times M$. Jinými slovy, každý z kanálů vstupu (číslo M) je konvoluován samostatně a hloubka (počet kanálů) se na výstupu vrstvy nemění. Finální výstup má rozměr $D_g \times D_g \times M$. Druhým krokem je bodová konvoluce, která je ve své podstatě lineární kombinací hodnot na stejných pozicích všech M matic. Filtry mají tvar $1 \times 1 \times M$, jejich počet ve vrstvě je číslo N. Z předchozího kroku zůstává rozměr D_g, počet výsledných matic bude N a tvar výstupu na konci hloubkově dělitelné konvoluce je $D_g \times D_g \times N$. Pokud vypočítáme, kolik operací součinu je nutno v těchto krocích provést v porovnání s běžnou konvolucí, je způsob hloubkově dělitelné konvoluce o až 90 % výpočetně úspornější. Zároveň data ukazují, že oproti podobně koncipované síti využívající běžnou konvoluci je přesnost jen zanedbatelně nižší [21][23][24].

3. Realizace

3.1 Datová sada pro učení modelu

Pro správnou volbu datové sady bylo nutné zvážit množství požadavků velmi specifických pro mojí práci. Zatímco základní podmínku snímků dopravních značek splňuje řada datových sad, příležitost ověřit přesnost modelu v praxi nám umožní jen hrstka. Značky se musejí podobat těm, které nalezneme v České republice, abychom mohli síť demonstrovat na datech vlastnoručně získaných v našem okolí. Tím ověříme, že je schopna pracovat i mimo výzkumnou hladinu uzavřených datových vzorků. Nakonec byla zvolena datová sada GTSRB (German Traffic Sign Recognition Benchmark) [29]. Sada je prací Institutu pro Neuroinformatiku Ruhrské univerzity v německém městě Bochum.

Jedná se o snímky německých dopravních značek, které se v drtivé většině případů shodují s českými ekvivalenty, což je přínosem pro aplikaci na naše lokální značení. Datová sada obsahuje téměř 40 000 příkladových obrázků na učení. Celkový počet snímků pak dosahuje okolo 50 000 (včetně části na ověření přesnosti modelu) s každým patřící do jedné ze 43 tříd – konkrétních dopravních značek. Třídy značek jsou nerovnoměrně rozděleny, například 12 různých tříd jsou značky zákazové, tedy ohraničené červeným kruhem. Podobnost instancí různých tříd pak pro modely učící se na sadě představují překážku. Vedle povahy dat a jejich objemu patří k přednostem sady GTSRB i jejich kvalita. Každý obrázek je unikátem, což je pro síť větším problémem a má na učení a následnou přesnost významný vliv [29].

Velkou nevýhodou datové sady je však velmi rozdílná frekvence zastoupení jednotlivých tříd. Zatímco některé ze značek (např. nejvyšší dovolená rychlost 50 km/h) jsou zastoupeny v tisícových počtech, jiné značky, jako například „Pozor chodci“, mají v souboru méně než 500 vzorků.

V příloze 1 je obsažen seznam s číslováním jednotlivých tříd značek, jejich originálními anglickými názvy a českými normovanými názvy.

Obrázek 13: značky přítomné v datové sadě GTSRB [Obr. 13]

3.1.1 Segmentace a příprava dat pro učení neuronové sítě

Ve svém výchozím formátu není sada GTSRB kompatibilní s knihovnou TensorFlow, kterou budeme pro tvorbu a učení modelu využívat. Hlavní překážku představuje formát obrázků, PPM (Portable Pixmap Format), který funkce pro zpracování obrazu knihovny nepodporují. Ve výchozím stavu je datová sada rozdělena do dvou částí, trénovací (učící) a testovací (ověřující). Učící část sady obsahuje přibližně 39 200 obrázků, ověřující část pak okolo 12 600. V dodávaném archivu je sada segmentována do podsložek, kde každá z nich reprezentuje jednu z tříd a obsahuje obrázky pouze z dané třídy. Pro naše využití zachováme strukturu složek, převedeme však všechny obrázky do formátu JPEG. K tomu byla využita knihovna jazyka Python pillow. Po tomto kroku byla celá datová sada připravena ke zpracování modulem TensorFlow.

Obrázek 14: ukázkové snímky z datové sady (vlastní tvorba)

3.2 Model konvoluční neuronové sítě

Klasifikace 43 tříd dopravních značek je z hlediska konvolučních neuronových sítí poměrně náročným úkolem. Jak již bylo popsáno v teorii, každá síť obsahuje velké množství vah, které je nutno iterativně měnit tak, aby bylo dosaženo přesného výstupu. Pokud bychom se rozhodli vyvinout, postavit a trénovat síť s nulovými zkušenostmi (váhy při tvorbě sítě vznikají jako náhodné hodnoty), bylo by velmi časově a výpočetně náročné dosáhnout uspokojivého výsledku. Z tohoto důvodu byl model sítě tvořen metodou Transfer Learning [31]. Ta spočívá ve využití již naučené sítě k podobné úloze, kdy se využije existujících hodnot parametrů a sníží se tím doba a náročnost učení.

Prostředím pro realizaci sítě byl zvolen programovací jazyk Python3 a jeho modul TensorFlow 2.8. Výhodou jazyka Python je velké množství modulů pro datovou analýzu a strojové učení, jednoduchý syntax a multiplatformní podpora. Vývoj probíhal v aplikaci Visual Studio Code [Soft. 1] a učení v prostředí Google Colab Pro [Soft. 4]. Samotná knihovna TensorFlow a její modely jsou kompatibilní s mobilními zařízeními, což přijde vhod pro tvorbu demonstrační aplikace. Knihovna nabízí pro případy Transfer Learningu velké množství obecně předtrénovaných modelů na sadě ImageNet [32]. Jejich výhoda spočívá v tom, že sada ImageNet obsahuje miliony obrázků, v případě funkčních modelů rozdělených do 1000 tříd. Model již tedy umí klasifikovat komplexní a značně odlišné předměty od koček až po lodě s relativně vysokou přesností. Náš výsledný model těchto zkušeností využije k dosažení co nejvyšší přesnosti klasifikace snímků dopravních značek.

3.2.1 Volba architektury modelu

Vzhledem k tomu, že finální nosičem naší sítě má být mobilní telefon, bylo k této skutečnosti nutno přihlédnout při volbě konkrétní architektury modelu. Pro tuto aplikaci nejsou vhodné modely typu VGG16 nebo VGG19, které obsahují více než 100 milionů parametrů [30]. Jejich velikosti dosahují stovek megabytů a čas výpočtu (inference time) by na mobilním zařízení byl v rámci stovek milisekund.

Na opačném konci spektra byly vytvořeny modely spadající do rodiny MobileNet, tedy konvoluční neuronové sítě určené pro mobilní zařízení s relativně nízkým výpočetním výkonem. Jejich rozměry jsou výrazně menší než u tradičních modelů, dosahují však přesnosti o maximálně jednotky procent nižší. Z této rodiny jsem volil i finální model, a to sice původní MobileNet [21] se 4,2 miliony parametry.

```
mobile = tf.keras.applications.mobilenet.MobileNet()
```

3.2.2 Úprava modelu

Na vstupu přijímá zvolený model barevné obrázky (3 barevné kanály) o rozměrech 224×224. Tvar vstupního tenzoru je tedy (224, 224, 3). Aby však bylo možné model trénovat pro klasifikaci našich pouze 43 tříd (na rozdíl od původních 1000), musela být klasifikační část upravena.

Úprava klasifikační části modelu spočívala v odstranění čtyř posledních vrstev, jejichž rozměr a naučené váhy odpovídaly původní datové sadě ImageNet. Za poslední průměrovou sdružovací vrstvou byla umístěna plně propojená vrstva s 43 výstupními neurony, každý pro jednu třídu dopravní značky. Aktivační funkcí poslední vrstvy byla zvolena funkce Softmax, protože jejím výstupem jsou hodnoty od 0 do 1. Ty vhodně reprezentují pravděpodobnost přítomnosti dopravní značky ve vstupních datech.

```
x = mobile.layers[-6].output
y = tf.keras.layers.GlobalAveragePooling2D(keepdims=False)(x)
output = Dense(units=43, activation="softmax")(y)
model = Model(inputs=mobile.input,outputs=output)
```

Detailní informace o výsledném modelu nám poskytla metoda summary() třídy model. Celkový počet parametrů sítě po úpravách činil 3272939.

```
model.summary()
```

V tabulce můžeme vidět kompletní architekturu sítě od vstupu až po výstupní vrstvu. Grafické znázornění architektury aplikací Netron [39] je dostupné v příloze 2.

Vrstva (typ vrstvy)	Tvar výstupního tenzoru vrstvy	Počet parametrů vrstvy
input_1 (Input Layer)	(None, 224, 224, 3)	0
conv1 (Conv2D)	(None, 112, 112, 32)	864
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)	288
conv_pw_1 (Conv2D)	(None, 112, 112, 64)	2048
conv_dw_2 (DepthwiseConv2D)	(None, 56, 56, 64)	576
conv_pw_2 (Conv2D)	(None, 56, 56, 128)	8192
conv_dw_3 (DepthwiseConv2D)	(None, 56, 56, 128)	1152
conv_pw_3 (Conv2D)	(None, 56, 56, 128)	16384
conv_dw_4 (DepthwiseConv2D)	(None, 28, 28, 128)	1152
conv_pw_4 (Conv2D)	(None, 28, 28, 256)	32768
conv_dw_5 (DepthwiseConv2D)	(None, 28, 28, 256)	2304
conv_pw_5 (Conv2D)	(None, 28, 28, 256)	65536
conv_dw_6 (DepthwiseConv2D)	(None, 14, 14, 256)	2304
conv_pw_6 (Conv2D)	(None, 14, 14, 512)	131072
conv_dw_7 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_pw_7 (Conv2D)	(None, 14, 14, 512)	262144
conv_dw_8 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_pw_8 (Conv2D)	(None, 14, 14, 512)	262144
conv_dw_9 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_pw_9 (Conv2D)	(None, 14, 14, 512)	262144
conv_dw_10 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_pw_10 (Conv2D)	(None, 14, 14, 512)	262144
conv_dw_11 (DepthwiseConv2D)	(None, 14, 14, 512)	4608
conv_pw_11 (Conv2D)	(None, 14, 14, 512)	262144
conv_dw_12 (DepthwiseConv2D)	(None, 7, 7, 512)	4608
conv_pw_12 (Conv2D)	(None, 7, 7, 1024)	524288
conv_dw_13 (DepthwiseConv2D)	(None, 7, 7, 1024)	9216
conv_pw_13 (Conv2D)	(None, 7, 7, 1024)	1048576
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1024)	0
dense (Dense)	(None, 43)	44075

Tabulka 1: architektura sítě, výstup modulu TensorFlow

3.2.3 Učení modelu

Abychom mohli neuronové síti dodat data z námi zpracované datové sady ve formátu, který odpovídá rozměrům vstupního tenzoru, obsahuje modul TensorFlow vestavěné funkce na úpravu obrazu a jeho převodu do kompatibilního matematického tvaru. První funkcí, kterou jsme využili k úpravě obrázků, bude ImageDataGenerator, jenž přeškáluje hodnoty snímků z celočíselných barevných hodnot na hodnoty v intervalu $< 0; 1 >$.

```
image_generator = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1/255)
```

Následně pomocí funkce flow_from_directory vygenerujeme v paměti objekt reprezentující sadu dat. Tato funkce vzorky rozdělí do tříd podle struktury složek, ve kterých se nacházejí. Jak bylo zmíněno v oddíle 3.1.1, každá podsložka reprezentuje jednu třídu. Funkce tuto strukturu zachová a zároveň k obrázkům pomocí ní vytvoří seznam správných výstupů. Byly vytvořeny 2 samostatné objekty dat, trénovací a validační. Validační sada využívá funkci flow_from_dataframe, protože správné výstupy pro validační část sady GTSRB jsou uloženy v tabulce formátu CSV.

```
test_data = image_generator.flow_from_directory(str(TRAIN_DIR),
target_size=INPUT_TENSOR_SHAPE, batch_size=32, shuffle=True)
val_data = image_generator.flow_from_dataframe(val_df, x_col="Filename", directory=VAL_DIR,
y_col="ClassId", target_size=INPUT_TENSOR_SHAPE, batch_size=32, shuffle=True)
```

Pro optimalizaci průběhu učení modelu bylo využito objektů Callbacks API Keras [51]. Ty umožňují ukládat data o průběhu učení sítě, stádia modelu na základě specifikovaných metrik či zasahovat do učení a měnit jednotlivé hyper-parametry. Callback TensorBoard zajišťuje ukládání dat pro pozdější zpracování aplikací TensorBoard, ReduceLROnPlateau sníží velikost učícího kroku, pokud metrika přesnosti na validační části přestane růst. EarlyStopping učení zastaví, pokud se výše zmíněná metrika nebude delší dobu měnit. ModelCheckpoint pak vytvoří na konci každé epochy soubor, ze kterého lze model zpětně načíst.

```
tensorboard_callbacks = tf.keras.callbacks.TensorBoard(LOG_DIR, histogram_freq=1)
dynamic_lr =
tf.keras.callbacks.ReduceLROnPlateau(monitor='val_accuracy',mode='max',min_delta=0.001,pati
ence=3,factor=0.5,verbose=1,cooldown=1,min_lr=0.00000001)
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',mode='max',
min_delta=0.0005,patience=5,verbose=1,restore_best_weights=True)
checkpoints = tf.keras.callbacks.ModelCheckpoint(WEIGHT_CHECKPOINT, mode="max",
monitor="val_accuracy", save_best_only=True, save_weights_only=True)
callbacks = [tensorboard_callbacks, dynamic_lr, early_stop, checkpoints]
```

Učení sítě probíhalo ve dvou fázích. První fáze učení se týkala pouze naší nové klasifikační části sítě a vrstev, které ji bezprostředně předcházejí. Druhá fáze byla učení celého modelu, kdy došlo k optimalizaci všech parametrů pro úlohu klasifikace dopravního značení. Prvním krokem bylo zmrazení vah všech vrstev sítě kromě posledních 23 [31][49][50]. Učení ovlivnilo pouze novou klasifikační část sítě a zmrazení části vah snížilo počet parametrů, který musel být během učení upraven, což zároveň zredukovalo časovou náročnost této fáze učení.

```
for layer in model.layers[:-23]:
    layer.trainable=False
```

Následně stačilo model zkompilovat – zmrazit. Kompilací modelu byla upevněna jeho aktuální konfigurace a definovány hyper-parametry učení. První fáze učení probíhala na 20 epoch, pro výpočet ztráty sloužila funkce Cross-entropy loss a optimalizátor parametrů využíval algoritmus Adam. Počáteční velikost učícího kroku byla 0,0001, aby nedošlo k narušení původních vah sady ImageNet velkou hodnotou gradientu chyby z naší nově inicializované klasifikační vrstvy.

```
model.compile(optimizer=Adam(lr=0.0001), loss=tf.keras.losses.CategoricalCrossentropy(),
metrics=["accuracy"])
steps_per_epoch = np.ceil(train_data.samples/train_data.batch_size)
model.fit(x=train_data, epochs=20,
steps_per_epoch=steps_per_epoch,validation_data=val_data,validation_steps=val_data.samples/
val_data.batch_size, callbacks=callbacks)
```

Ve druhé fázi učení sítě došlo k odmrazení zbytku modelu, bylo tedy možné učit celou síť se všemi parametry. Byly také načteny váhy modelu s nejlepším výsledkem přesnosti na validační sadě z předchozí fáze učení. Dostatečná přesnost nové klasifikační vrstvy umožnila učit celý model bez negativního vlivu na původní váhy horní části sítě.

```
model.load_weights(WEIGHT_CHECKPOINT)
for layer in model.layers:
  layer.trainable=True
```

Model sítě byl opět zkompilován a učen na 40 epoch s velikostí učícího kroku 0,0001.

```
model.compile(optimizer=Adam(lr=0.0001), loss=tf.keras.losses.CategoricalCrossentropy(),
metrics=["accuracy"])
model.fit(x=train_data, epochs=40,
steps_per_epoch=steps_per_epoch,validation_data=val_data,validation_steps=val_data.samples/
val_data.batch_size, callbacks=callbacks)
```

3.3 Validace naučeného modelu

Za účelem posouzení výkonnosti sítě mimo trénovací podmnožinu slouží validační (ověřovací) podmnožina datové sady GTSRB. Ta čítá okolo 12 600 obrázků náhodných tříd. Knihovna TensorFlow nabízí vestavěnou funkci na validaci, pomocí které zároveň vypočítá hodnotu přesnosti během validačních kroků. Validace byla provedena jak po konci každé z trénovacích epoch, tak i na konci každé trénovací fáze pomocí nejlepších obnovených parametrů sítě.

```
model.load_weights(WEIGHT_CHECKPOINT)
score = model.evaluate(x=val_data, batch_size=val_data.batch_size,
steps=val_data.samples/val_data.batch_size)
print("Loss: ", score[0], "Accuracy: ", score[1])
```

Další validační metrikou sítě byly hodnoty precision a recall, v našem případě vážené vůči frekvenci výskytu daných tříd (některé třídy jsou početnější než ostatní).

```
import sklearn.metrics as metrics
print("Weighted precision score is: ",
metrics.precision_score(val_data.classes,preds_filtered,average="weighted"))
print("Weighted recall score is: ",
metrics.recall_score(val_data.classes,preds_filtered,average="weighted"))
```

Poslední statistikou využitou pro posouzení přesnosti sítě byla matice zmatení vytvořená knihovnou seaborn.

```
import seaborn as sns
cm = tf.math.confusion_matrix(
    val_data.classes, preds_filtered, num_classes=43)
fig, ax = plt.subplots(figsize=(10, 10))
ax = sns.heatmap(cm, cbar=False, cmap="Blues", linewidths=1, linecolor='grey')
ax.plot()
plt.show()
```

3.4 Praktická zkouška sítě na vlastních datech

Pro účel zkoušky sítě na čistě vlastních datech, jež nejsou součástí sady GTSRB, byly zvoleny 3 snímky. Jedná se o snímky, které budou pro síť představovat větší překážku, než ořezané a vycentrované snímky z původní sady. Ke zpracování a přípravu snímku pro model neuronové sítě nám slouží metoda využívající modul image knihovny TensorFlow [33].

```
from tensorflow import image as tf_image
def predict_image(test_img):
    test_img = prepare_image(test_img)
    test_preds = model.predict(test_img)
    print("Class was identified as:", np.argmax(test_preds, axis=1)
        [0], "with confidence of:", int(100*(test_preds.max())),"%")
    print(label_map[(np.argmax(test_preds, axis=1))[0]])
test_images = ["hlavni.jpeg","kruhac.jpg","zakazjednosmerka.jpg"]
for image in test_images:
    predict_image(image)
```

3.5 Konverze modelu do formátu pro mobilní zařízení

Po naučení modelu konvoluční neuronové sítě bylo nutné převést ho do formátu, který lze implementovat v mobilní aplikaci. Pro použití modelů neuronových sítí na mobilních zařízení obsahuje modul TensorFlow verzi TensorFlow Lite (dále jen TF Lite). TF Lite umožňuje převádět modely do formátu tflite, implementovat je do aplikací a optimalizovat jejich běh podle zařízení, aby bylo dosaženo co nejlepšího výkonu. Náš naučený model budeme převádět do formátu tflite, během konverze ale nebudeme provádět žádné optimalizace. Za účelem snížení výpočetní náročnosti modelu lze během konverze změnit číselnou povahu z 32bitových desetinných čísel (float32) na 8bitová celá čísla (Int8), dochází tím však ke zhoršení přesnosti sítě [52]. I náš Lite model bude tedy využívat 32bitových desetinných čísel a konverze tím nijak nevlivní jeho výkon.

O samotnou konverzi modelů mezi formáty se stará třída tf.lite.TFLiteConverter knihovny TensorFlow. Převádět budeme z formátu tf.Keras.Model pomocí metody from_keras_model(). Celý kód převodu vypadá následovně:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
with tf.io.gfile.GFile(OUT_LITE_MODEL, 'wb') as f:
  f.write(tflite_model)
```

Výsledkem konverze je soubor model.tflite, který obsahuje naučený model ve formátu použitelném knihovnou TF Lite.

3.6 Demonstrační mobilní aplikace

Zvoleným demonstračním nosičem naší neuronové sítě je aplikace pro chytrý mobilní telefon. Příčinou rozhodnutí je skutečnost, že moderní mobilní zařízení nabízejí poměrně vysoký výpočetní výkon, dostatečně kvalitní fotoaparát (zdroj obrazových dat) a vývoj aplikací není časově ani technologicky náročný. Je to také jeden z důvodů, proč byla k vývoji sítě zvolena knihovna TensorFlow. Modely ve formátu TensorFlow Lite jsou podporovány napříč programovacími jazyky a platformami.

3.6.1 Volba cílové platformy

Jako cílová platforma byl zvolen operační systém iOS společnosti Apple. Oproti ostatním platformám má řadu výhod, pro nás je však klíčová jednoduchost vývoje, podpora modelů TensorFlow Lite nativním jazykem platformy Swift a velmi vysoký výpočetní výkon. Hardware společnosti Apple spolu s vlastním frameworkem Core ML [34] pro strojové učení tvoří ideální prostředí pro nasazení modelů na mobilních zařízeních.

3.6.2 Demonstrační zařízení

Zařízení, jenž bude využito k provozu aplikace je mobilní telefon Apple iPhone 13. Dostatečný výkon našeho modelu zajistí chipset Apple A15 Bionic [35]. Ten obsahuje 6jádrový procesor (CPU), 4jádrovou grafickou kartu (GPU) a 16jádrový Neural Engine (NPU – neurální výpočetní jednotku). Díky jejich kombinaci s frameworkem Core ML bude náš model moci klasifikovat značky v reálném čase s velmi malým časem výpočtu (inference time).

3.6.3 Tvorba aplikace

Jako základ aplikace byla použita volně dostupná příkladová aplikace od tvůrců modulu TensorFlow [36]. Ta nabízí základní uživatelské rozhraní, jenž nám umožní sledovat 3 nejpravděpodobnější klasifikace objektu uprostřed záběru ze zadní kamery. Zároveň informuje o samotném modelu, vidíme rozměr vstupního obrazu a dobu výpočtu. Ve výchozím stavu využívá aplikace pouze procesor telefonu a můžeme si navolit, na kolika výpočetních vláknech může model provádět výpočty. Klasifikace objektů probíhá v reálném čase a je zobrazena zároveň s obrazem fotoaparátu.

I v základním stavu je aplikace přímo použitelná pro naši úlohu. Kód je psán formou plug-and-play pro jakékoliv klasifikační modely ve formátu TensorFlow Lite. Stačilo by do aplikace dosadit soubor našeho modelu a textový soubor se štítky dopravních značek v pořadí

od 1 do 43 podle tříd. Pak by již aplikace fungovala jako plnohodnotný klasifikátor dopravních značek

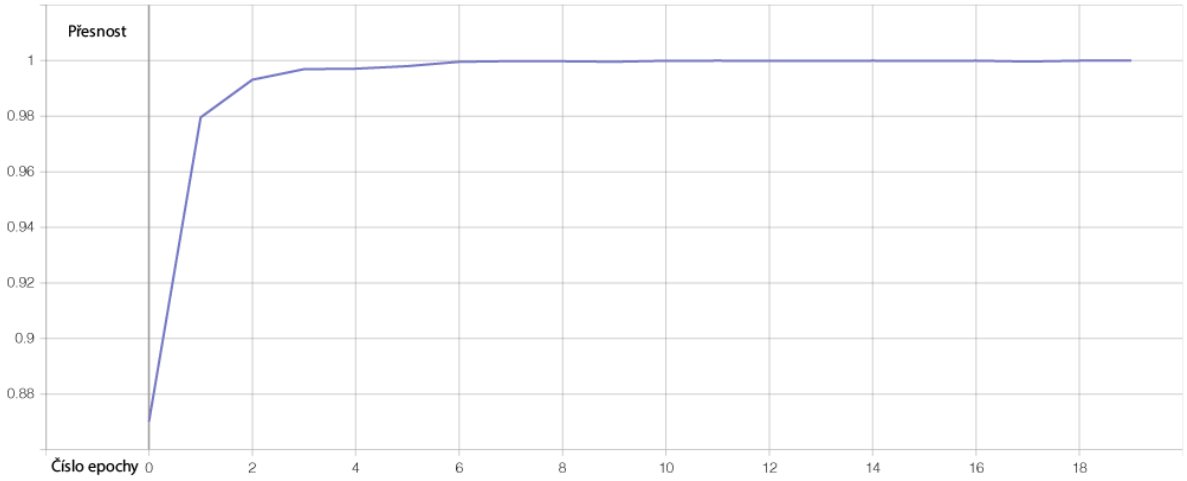
Za účelem optimalizace chodu a zlepšení uživatelského prostředí bylo však nutné provést několik změn. První změnou bylo využití akcelerátoru Neural Engine. Knihovna TensorFlow Lite pro programovací jazyk Swift nabízí funkci CoreMLDelegate(), která po implementaci spouští všechny podporované operace neuronové sítě na Neural Enginu [37]. CoreMLDelegate() resp. framework Core ML se automaticky stará o alokaci a využití výpočetního výkonu, bez zásahů uživatele. Spouštění na akcelerátoru by mělo přinést nejen zlepšení výkonu aplikace, ale taky snížení její energetické náročnosti. Provoz daných operací na dedikovaném akcelerátoru je vždy výrazně účinnější než na všestranné výpočetní jednotce, jakou je procesor [38].

Změny uživatelského prostředí spočívaly ve tvorbě vlastní ikony aplikace, změny úvodní obrazovky a odstranění možnosti měnit počet výpočetních vláken. K potřebným úpravám aplikace sloužilo vývojové prostředí Xcode [Soft. 3] společnosti Apple. Pro tvorbu a úpravu grafických aktiv byla použita aplikace Adobe Illustrator [Soft. 2]. Testovací spuštění aplikace lze v prostředí Xcode mimo připojeného iPhonu či iPadu provést i na počítači Mac s architekturou procesoru Apple M1.

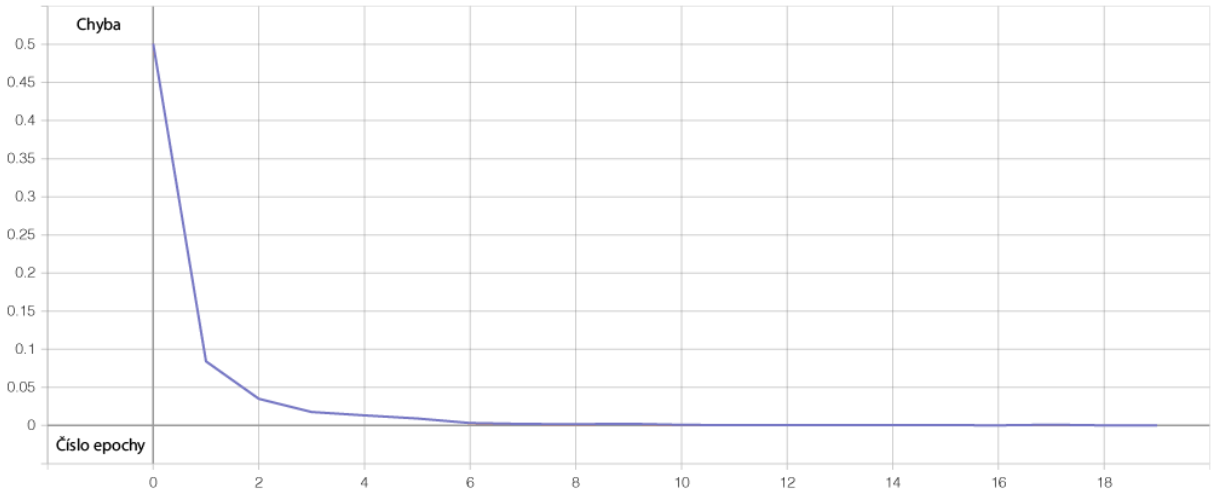
4. Výsledky

4.1 Průběh učení neuronové sítě

Po 20 trénovacích epochách první fáze učení dosáhla síť na uzavřeném trénovacím souboru dat přesnosti 100 % a hodnoty ztrátové funkce $2{,}3835 \times 10^{-4}$. Nejlepší výsledek na validační sadě, ze něhož byly načteny parametry pro další fázi učení, činil 0,2219 pro hodnotu ztrátové funkce s přesností 94,39 %. Velikost učícího kroku na konci poslední epochy klesla na $1{,}25 \times 10^{-5}$ z původních 10^{-4}. Statistiky ukládané v průběhu učení byly vizualizovány aplikací TensorBoard do grafů. První část učení trvala ve výpočetním prostředí Google Colab Pro s GPU akcelerací přibližně 35 minut.

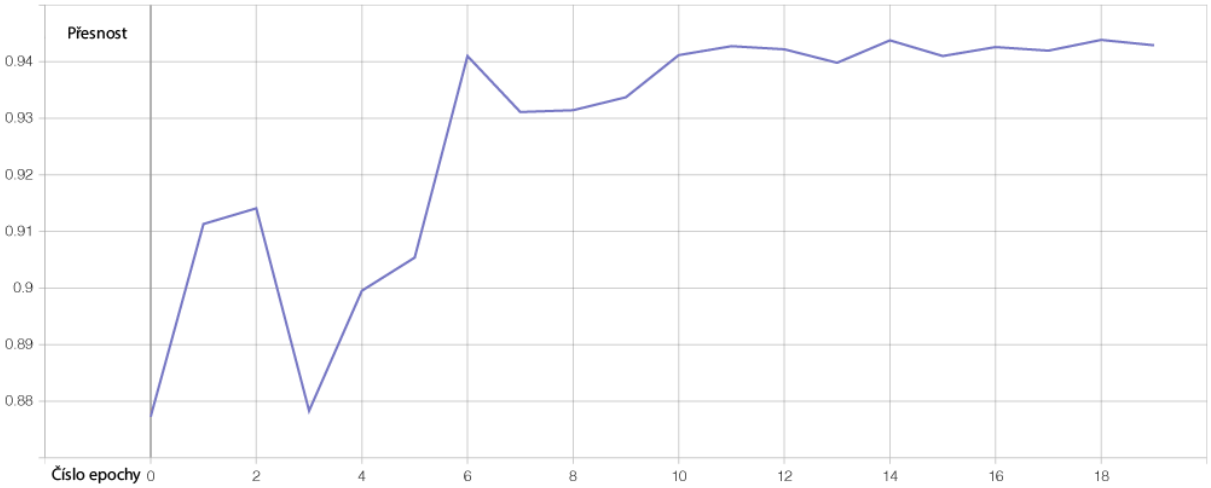


Obrázek 15:graf vývoje přesnosti sítě na učící složce – 1. fáze učení (vlastní tvorba, nevyhlazeno)

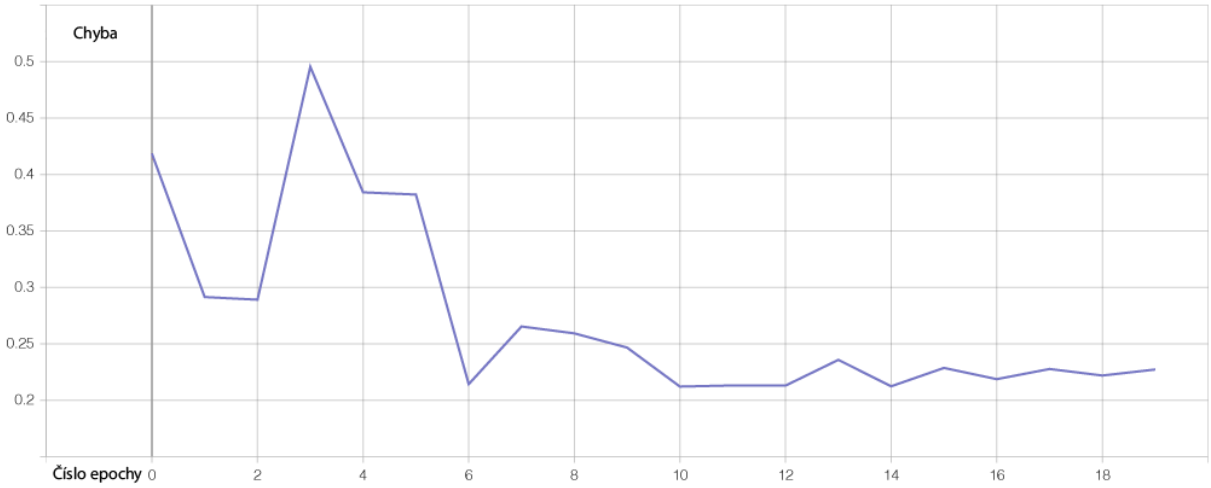


Obrázek 16: graf vývoje hodnoty ztrátové funkce sítě na učící složce – 1. fáze učení (vlastní tvorba, nevyhlazeno)

Na grafu ztrátové funkce trénovací sady můžeme pozorovat, že již od první epochy byla její hodnota poměrně nízká. Rychle konvergovala a později se již výrazně neměnila. To samé platí o přesnosti sítě v následujícím grafu, která byla již od začátku poměrně vysoká a rychle dosáhla přibližně 100 %.



Obrázek 17: graf vývoje přesnosti sítě na validační složce – 1. fáze učení (vlastní tvorba, nevyhlazeno)



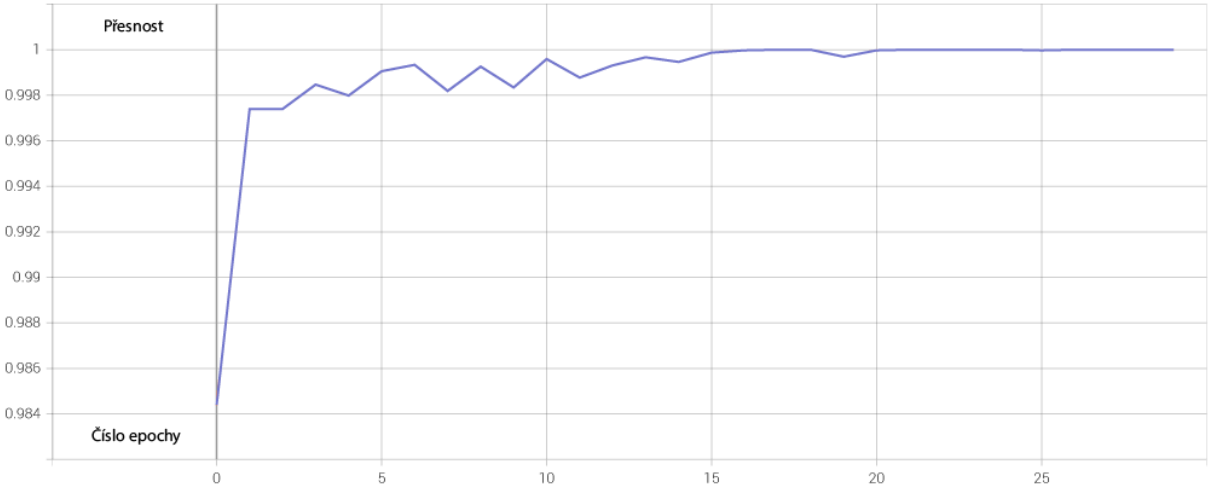
Obrázek 18: graf vývoje hodnoty ztrátové funkce na validační složce – 1. fáze učení (vlastní tvorba, nevyhlazeno)

I když se vysoká přesnost sítě může jevit jako úspěch, značí také, že velmi pravděpodobně došlo k mírnému přeučení. To potvrzují i grafy vývoje hodnot ztrátové funkce a přesnosti na validační složce datové sady GTSRB, jejichž tvary hrubě neodpovídají těm pro trénovací část sady. Ukazatelem přeučení je i rozdíl mezi přesnostmi na trénovací a validační sadě. Ten byl na konci první fáze učení poměrně vysokých 5,61 %.

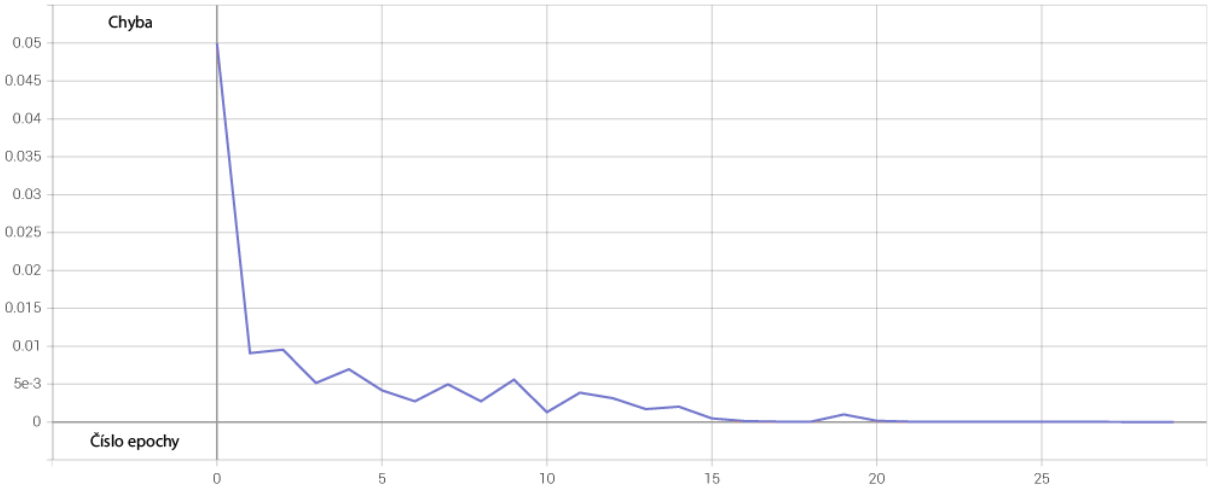
Před spuštěním druhé fáze učení byla provedena finální evaluace sítě na validační složce pomocí nejlepších dosažených parametrů.

```
394/394 [==============================] - 46s 115ms/step - loss: 0.2219 - accuracy: 0.9439
Loss:  0.22193755209445953  Accuracy:  0.9438638091087341
```

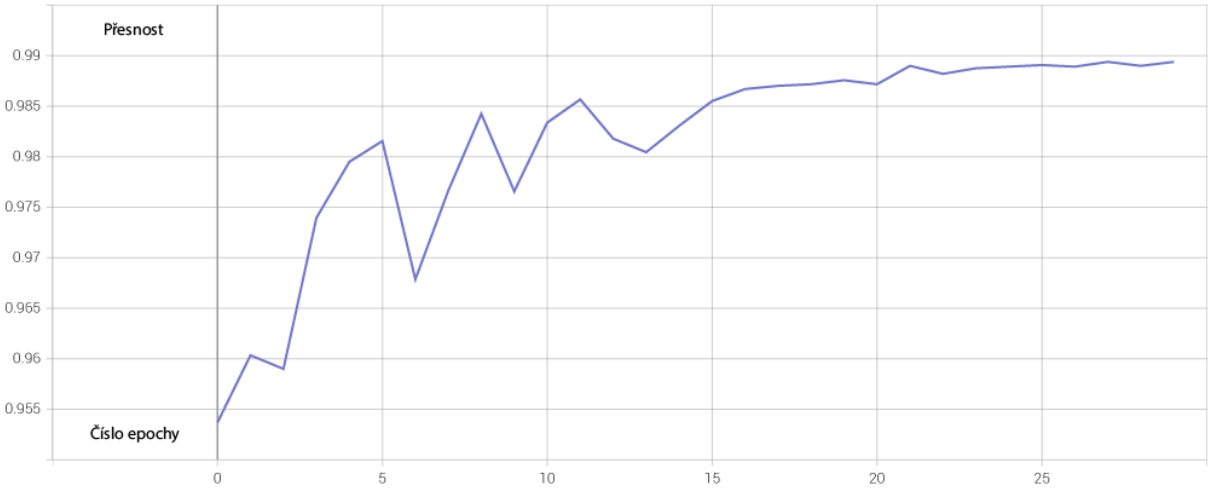
Druhá fáze učení měla trvat celkem 40 epoch včetně validační zkoušky na konci každé z nich. Byla však předčasně ukončena na konci 30. epochy. Nejlepší výsledek na validační složce, který byl následně obnoven, dosahoval hodnot ztrátové funkce 0,0403 a přesnosti 98,95 %. Přesnost na trénovací složce byla opět 100 % a hodnoty ztrátové funkce $3{,}2853 \times 10^{-6}$. Učící krok poklesl z 10^{-4} na $6{,}25 \times 10^{-6}$ na konci poslední epochy. Druhá fáze učení modelu vyžadovala zhruba 2 hodiny ve výpočetním prostředí Google Colab Pro s GPU akcelerací.



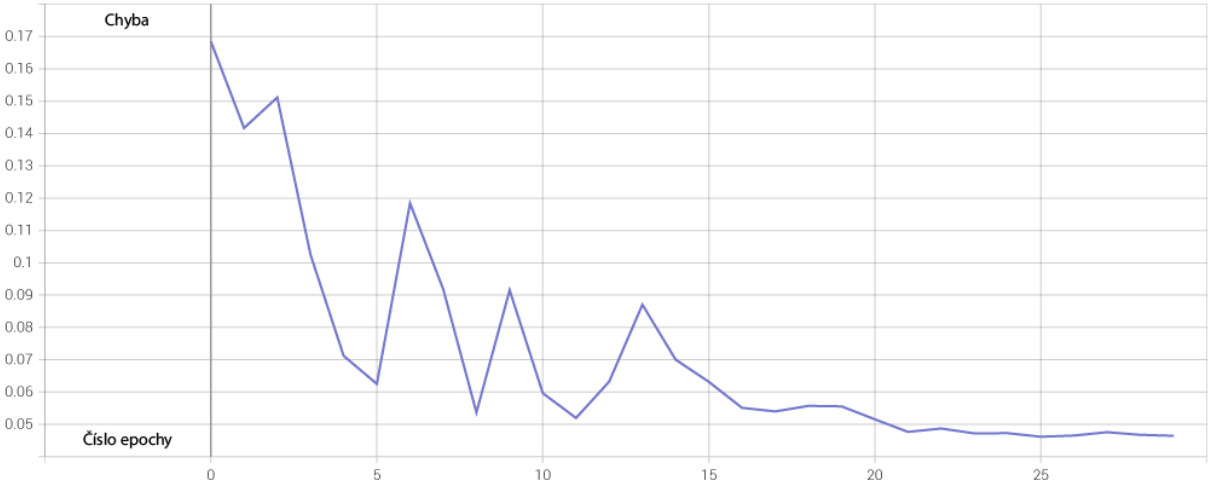
Obrázek 19: graf vývoje přesnosti sítě na učící složce – 2. fáze učení (vlastní tvorba, nevyhlazeno)



Obrázek 20: graf vývoje hodnoty ztrátové funkce sítě na učící složce – 2. fáze učení (vlastní tvorba, nevyhlazeno)



Obrázek 21: graf vývoje přesnosti sítě na validační složce – 2. fáze učení (vlastní tvorba, nevyhlazeno)



Obrázek 22: graf vývoje hodnoty ztrátové funkce na validační složce – 2. fáze učení (vlastní tvorba, nevyhlazeno)

I v případě druhé fáze došlo k mírnému přeučení, avšak výrazně menšímu než v první fázi. Rozdíl mezi přesnostmi činil pouhých 1,05 % a model dosáhl obecně vysoké přesnosti na validační sadě. Po ukončení učení a načtení nejlepších parametrů byla opět provedena validace.

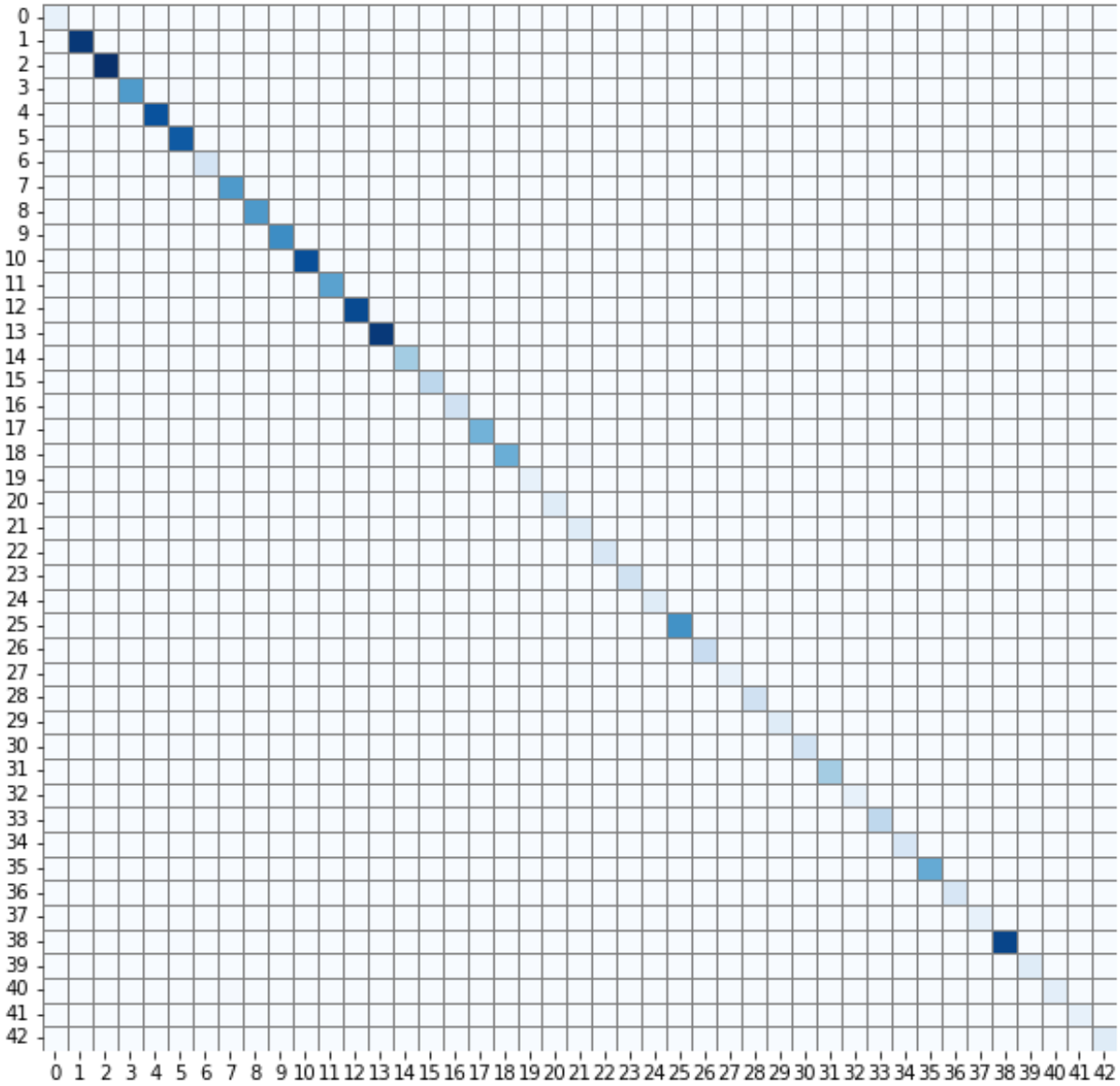
```
394/394 [==============================] - 46s 115ms/step - loss: 0.0403 - accuracy: 0.9895
Loss: 0.040345340967178345 Accuracy: 0.9895486831665039
```

4.2 Ověření výkonu neuronové sítě

Další informace o výkonu modelu sítě poskytují vážené statistiky precision a recall, které jsou obě téměř shodné s hodnotou přesnosti, indikující velmi dobrou celkovou přesnost sítě.

```
Weighted precision score is: 0.9897391983654442
Weighted recall score is: 0.9895486935866984
```

Poslední reprezentací úspěšnosti sítě je matice zmatení.



Obrázek 23: matice zmatení pro celou validační část sady GTSRB (vlastní tvorba)

Z matice můžeme jako vedlejší efekt pozorovat nerovnoměrnost četností tříd v datové sadě GTSRB. Políčka (správně klasifikovaných) četnějších tříd jsou výrazně jasnější než tříd s nižším počtem jednotlivých snímků. Všechna barevná pole leží na diagonále matice, poukazující na velmi vysokou přesnost klasifikace.

4.3 Výsledky praktické zkoušky sítě na vlastních datech

Model neuronové sítě byl vyzkoušen na třech snímcích dopravních značek, které nebyly součástí datové sady GTSRB. Nejednalo se o vystředěné a ořezané snímky značek jako v datové sadě, ale o snímky obsahující i jiné objekty a pozadí.

Obrázek 24: fotografie značky Kruhový objezd [Obr. 24]

Obrázek 25: fotografie značky Zákaz vjezdu všech vozidel [Obr. 25]

Obrázek 26: fotografie značky Hlavní pozemní komunikace [Obr. 26]

Výstup kódu pro ověření přesnosti indikoval číselné označení předpovězené třídy, její anglický název a procentuální hodnotu jistoty, se kterou byla určena. České názvy lze odvodit podle tabulky v příloze 1. Všechny snímky byly úspěšně klasifikovány.

```
Class was identified as: 40 with confidence of: 95 %
turn_circle
Class was identified as: 17 with confidence of: 98 %
no_way_one_way
Class was identified as: 12 with confidence of: 95 %
right_of_way_general
```

4.4 Mobilní aplikace

4.4.1 Výpočetní výkon a energetická náročnost aplikace

Během vývoje aplikace se podařilo splnit všechny vytyčené změny. Během převodu akcelerátoru neuronové sítě z procesoru na Neural Engine bylo provedeno měření pomocí původní funkce aplikace na měření doby výpočtu.

Akcelerátor	Doba výpočtu (průměrná)
Procesor (1 vlákno)	16.2 ms
Procesor (6 vláken)	9.7 ms
Neural Engine (Core ML)	2.5 ms

Tabulka 2: porovnání akcelerátorů neuronových sítí na chipsetu A15 Bionic

Použitím Neural Enginu jako hlavní výpočetní jednotky pro běh naší neuronové sítě bylo možné docílit přibližně 6,5krát lepšího výkonu modelu oproti běhu na 1 výpočetním vlákně procesoru. Díky době výpočtu můžeme vypočítat i počet snímků, jenž dokáže model za sekundu klasifikovat.

$$počet\ snímků\ za\ sekundu = \frac{1000\ ms}{doba\ výpočtu\ v\ ms}$$

Dosazením do rovnice zjišťujeme, že je model pomocí akcelerace na Neural Enginu schopný klasifikovat průměrně 400 snímků za sekundu (400 FPS).

Před změnou akcelerátoru využívala aplikace se 6 výpočetními vlákny zhruba 30 % výkonu procesoru. Po změně akcelerátoru na Neural Engine aplikace využívá zhruba 10 % výkonu procesoru a dle interních metrik aplikace Xcode dosahuje přibližně poloviční energetické náročnosti.

4.4.2 Uživatelské rozhraní aplikace

Aplikaci byla vytvořena nová ikona, úvodní obrazovka a bylo odstraněno logo TensorFlow Lite uvnitř hlavní obrazovky aplikace.

Obrázek 27: nová ikona aplikace (vlastní tvorba)

Obrázek 28: úvodní obrazovka aplikace (vlastní tvorba)

Obrázek 29: hlavní obrazovka aplikace (vlastní tvorba)

4.4.3 Klasifikační výkon aplikace

Zkoušky v reálném provozu byly provedeny v mém bezprostředním okolí na dostupném dopravním značení.

Obrázek 30: klasifikace reálné značky v provozu

Obrázek 31: klasifikace reálné značky v provozu

Obrázek 32: klasifikace reálné značky v provozu

Pokud se nacházíme v dostatečné blízkosti značky a držíme zařízení rovnoběžně s dopravní značkou, lze dosáhnout úspěšné a velmi přesné klasifikace. S rostoucí vzdáleností

a úhlem však jistota klasifikace klesá až přestane být jednotná a výstup je pouze šum nízkých pravděpodobností přítomností náhodných značek.

Obrázek 33: klasifikace reálné značky v provozu

5. Diskuze

Z výsledků učení sítě je vidět, že její výkon v úloze klasifikace dopravního značení je velmi dobrý. Síť během několika málo epoch dosáhla přesnosti nad 99 %, následně rovných 100 % na trénovací složce, a při ověření sítě na validační složce datové sady byla velmi vysoká přesnost potvrzena hodnotou 98,95 %. Síť si vedla velmi dobře i s vzorky, jež nebyly součástí sady GTSRB. V kontextu konvolučních neuronových sítí typu MobileNet, které byly naučeny na datové sadě GTSRB, se jedná o extrémně vysoký výsledek. V porovnání s dalším pokusem využívajícím stejnou architekturu [Mirek Stanek, 33, 43] má náš model o 30,63 % vyšší přesnost (Stankův model dosáhl přesnosti 68,32 % na validační složce datové sady). Příčinou může být rozdílný postup při tvorbě modelu. Stankův model byl tvořen pomocí modelové knihovny TensorFlow Hub, jejíž modely nenabízejí tak vysokou flexibilitu. Nebylo tak pravděpodobně možné provést jemné optimalizace vrstvu po vrstvě jako v případě našeho funkčního modelu. Také učení jeho modelu probíhalo pouze v jedné fázi bez optimalizace průběhu a použití callbacků.

Pokud bychom však rozšířily naše porovnání na sítě, které jsou postaveny na odlišné architektuře, lze nalézt výrazně výkonnější modely s přesností okolo 99,7 % [44][45]. Ty však nemusejí nutně být optimalizovány pro běh na zařízeních s nižším výpočetním výkonem. Nejblíže k našemu modelu je síť MicroNet, jež dosáhla přesnosti 98,9 % [47]. Její velikost i počet parametrů jsou překvapivě výrazně menší než u naší, již poměrně kompaktní sítě a dosahuje pouze o 0,05 % nižší přesnosti. Naše síť by se v globálním měřítku přesnosti na datové sadě GTSRB díky převaze nad sítí MicroNet [47] zařadila na 4. místo.

Dozajista největší limitací při učení našeho modelu byla datová sada GTSRB, resp. její výchozí forma. Jak bylo již zmíněno, zastoupení jednotlivých tříd značek se značně liší, v některých případech i téměř 10krát. To při učení sítě způsobuje předpojatost (bias) vůči zastoupenějším třídám. Optimalizační algoritmus se bude častěji snažit upravit parametry sítě tak, aby správně klasifikovala početnější třídy než ty početně znevýhodněné. Při tvorbě sítí v produkčním prostředí se data zpravidla augmentují [46] tak, aby bylo jejich zastoupení rovnoměrné a bylo jich dostatečné množství. Jedná se však o časově, výpočetně a prostorově (z hlediska úložiště) náročný proces, a nebyl proto zařazen do rámce této práce. Učení na původní sadě nabízí dobrou metriku pro porovnání s ostatními sítěmi, u kterých nebyla augmentace implementována.

Při tvorbě mobilní aplikace se podařilo realizovat všechny vytyčené změny uživatelského prostředí a aplikaci maximálně optimalizovat pro co nejrychlejší běh sítě a energetickou účinnost. Použitím akcelerátoru Neural Engine bylo možné provádět výpočty téměř 6,5krát rychleji a s přibližně poloviční energetickým dopadem. Samotná klasifikační schopnost aplikace však zůstává omezena základní povahou modelu, resp. datové sady GTSRB. Model je naučený poznávat jen 43 dopravních značek, což je zlomek ze všech, které na českých silnicích najdeme. Zároveň také nedokáže rozeznat značku z větší vzdálenosti či pod větším úhlem, protože sada GTSRB obsahuje pouze přiblížené a ořezané snímky značek bez pozadí, pořízené téměř kolmo na rovinu značky. V rámci naší práce tak nelze bohužel statisticky kvantifikovat výkon sítě v demonstrační aplikaci, protože faktory úhlu a vzdálenosti od dopravní značky, hrající významnou roli, nelze v praxi přesně měřit. Veškeré hodnocení lze tedy provést jen slovně. Bohužel není ani možné pomocí zdánlivého klasifikačního výkonu a statistik (procenta jistoty) z aplikace tvořit obecnější závěry o výkonu sítě v mobilním formátu.

Pokud by měla být maximalizována užitná hodnota aplikace v praxi, bylo by nutné učit model na značně augmentovaných datech, která by dosahovala dostatečné rovnoměrnosti zastoupení tříd a objemu (v rámci set tisíců snímků). Pravděpodobně by model také musel být učen pro detekci, nikoliv pouze klasifikaci. Detekce přidává ke klasifikaci snímku také přesnou lokalizaci dané dopravní značky v celém obraze. Síť by tak dokázala značky vždy nalézt a označit bez problémů spojených s přítomností pozadí, faktoru vzdálenosti, úhlu a ostatních objektů v záběru.

6. Závěr

Za cíl této práce byl stanoven návrh, tvorba a realizace modelu konvoluční neuronové sítě pro klasifikaci dopravního značení z datové sady GTSRB. Součástí práce byla i praktická demonstrace formou mobilní aplikace, jenž klasifikuje obrazová data v reálném čase. Analyzovali jsme datovou sadu, zvolili vhodnou architekturu modelu z knihovny TensorFlow a pomocí připravených dat ho naučili metodou Transfer Learning snímky klasifikovat. Při učení modelu sítě bylo dosaženo velmi vysoké přesnosti, která byla následně potvrzena i na validační části datové sady. K analýze přesnosti nám posloužily hodnoty ztrátové funkce, přesnosti, hodnota recall, precision a graf matice zmatení. Model si vedl dobře i se snímky, které nebyly součástí původní datové sady a úspěšně klasifikoval jejich obsah. Potom byla provedena jeho konverze do formátu TensorFlow Lite kompatibilního s mobilními zařízeními. Jako základ demonstrační aplikace posloužila příkladová aplikace od tvůrců knihovny TensorFlow. Ta byla upravena pro optimalizaci běhu sítě použitím dedikovaného akcelerátoru pro strojové učení, čímž bylo dosaženo nižší energetické náročnosti a rychlejšího běhu aplikace. Změn doznalo i uživatelské prostředí aplikace, které bylo upraveno v souladu s úlohou klasifikace dopravního značení. Při praktických zkouškách aplikace byl demonstrován její velmi dobrý výkon a poměrně vysoká přesnost klasifikace, která je však značně omezena prostředím provozu na pozemních komunikacích. Tím došlo k odhalení slabin modelu naší neuronové sítě a vytvoření prostoru pro její další zlepšení. Do budoucna by mohl být model přetrénován pomocí augmentovaných dat či vytvořen od základů na augmentovaných datech pro detekci, nikoliv jen klasifikaci snímků. To by zaručilo jeho ideální výkon ve skutečném provozu a v případech jako např. pohyb kamery umístěné v automobilu.

7. Reference

7.1 Informační zdroje

1. BURNS, Ed. DEFINITION machine learning. *TechTarget|SearchEnterpriseAI* [online]. TechTarget, 2021, Březen 2021 [cit. 2022-01-08]. Dostupné z: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

2. Machine Learning. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-, 9. ledna 2022 [cit. 2022-01-09]. Dostupné z: https://en.wikipedia.org/wiki/Machine_learning

3. ASIRI, Sidath. Machine Learning Classifiers. *Towards Data Science* [online]. Kanada: Medium, 2018, 11. července 2018 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623

4. LECUN, Yann a Christopher J.C. CORTES. THE MNIST DATABASE of handwritten digits. *Yann LeCun* [online]. New York: The Courant Institute of Mathematical Sciences, New York University, 1998, Listopad 1998 [cit. 2022-01-08]. Dostupné z: http://yann.lecun.com/exdb/mnist/

5. JACOBSON, Lee. Introduction to Artificial Neural Networks Part 2 - Learning. *TheProjectSpot* [online]. Bristol, Spojené Království: theProjectSpot, 2014, 26. března 2014 [cit. 2022-01-08]. Dostupné z: https://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-2-learning/8

6. PATEL, Dhaval. Machine Learning Tutorial Python: 9 - Decision Tree. *Youtube* [online]. San Bruno, California: Google, 2018, 17 listopadu 2018 [cit. 2022-01-09]. Dostupné z: https://www.youtube.com/watch?v=PHxYNGo8NcI&list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw&index=11

7. PEIXEIRO, Marco. The Complete Guide to Support Vector Machine (SVM). *Towards Data Science* [online]. Kanada: Medium, 2019, 29. července 2019 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/the-complete-guide-to-support-vector-machine-svm-f1a820d8af0b

8. PEIXEIRO, Saishruthi. Linear Regression — Detailed View. *Towards Data Science* [online]. Kanada: Medium, 2019, 29. července 2019 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86

9. DEEPANSHI. All you need to know about your first Machine Learning model – Linear Regression. *Analytics Vidhya* [online]. Bengaluru: Analytics Vidhya, 2021, 25. května 2021 [cit. 2022-01-09]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-your-first-machine-learning-model-linear-regression/

10. KAUSHIK, Saurav. An Introduction to Clustering and different methods of clustering. *Analytics Vidhya* [online]. Bengaluru: Analytics Vidhya, 2016, 3. listopadu 2016

[cit. 2022-01-09]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-your-first-machine-learning-model-linear-regression/

11. [AL-MA'AMARI, Mohammed. Deep Neural Networks for Regression Problems. *Towards Data Science* [online]. Kanada, 2018, 29. září 2018 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/deep-neural-networks-for-regression-problems-81321897ca33

12. IBM CLOUD EDUCATION. Neural Networks. *IBM.com* [online]. Armonk, New York State: IBM.com, 2020, 17. srpna 2020 [cit. 2022-01-08]. Dostupné z: https://www.ibm.com/cloud/learn/neural-networks

13. SHARMA, Sagar. Activation Functions in Neural Networks: Sigmoid, tanh, Softmax, ReLU, Leaky ReLU EXPLAINED !!! *Towards Data Science* [online]. Kanada: Medium, 2017, 6. září 2017 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

14. MCGONAGLE, John, George SHAIKOUSKI a Christopher WILLIAMS. Backpropagation. *Brilliant.org* [online]. San Francisco: Brilliant Worldwide [cit. 2022-01-08]. Dostupné z: https://brilliant.org/wiki/backpropagation/

15. SHARMA, Sagar. What the Hell is Perceptron?: The Fundamentals of Neural Networks. *Towards Data Science* [online]. Kanada: Medium, 2017, 9. září 2017 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

16. SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. *Towards Data Science* [online]. Kanada: Medium, 2018, 15. prosince 2018 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

17. VUDMOULIN a FVISION. Convolution arithmetic. *GitHub.com* [online]. San Francisco: GitHub, 2019, 12. dubna 2019 [cit. 2022-01-08]. Dostupné z: https://github.com/vdumoulin/conv_arithmetic

18. IBM CLOUD EDUCATION. Convolutional Neural Networks. *IBM.com* [online]. Armonk, New York State: IBM.com, 2020, 20. října 2020 [cit. 2022-01-08]. Dostupné z: https://www.ibm.com/cloud/learn/convolutional-neural-networks

19. AGARWAL, Rahul. An End to End Introduction to GANs: Easy Peasy Lemon Squeezy. *Towards Data Science* [online]. Kanada: Medium, 2019, 19. července 2019 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/an-end-to-end-introduction-to-gans-bf253f1fa52f

20. LECUN, Yann, Leon BOTTOU, Yoshua BENGIO a Patrick HAFFNER. *Gradient-Based Learning Applied to Document Recognition* [online]. Québec, Canada, 1998 [cit. 2022-01-09]. Dostupné z: http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf. Vědecký článek. Université de Montréal.

21. HOWARD, Andrew G., Menglong ZHU, Bo CHEN, Dmitry KALENICHENKO, Weijun WANG, Tobias WEYAND, Marco ANDREETTO a Hartwig ADAM. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv:1704.04861v1* [online]. Ithaca, New York State: arXiv, 2017, 17. dubna 2017 [cit. 2022-01-09]. Dostupné z: https://arxiv.org/pdf/1704.04861.pdf

22. CODEEMPORIUM. Depthwise Separable Convolution - A FASTER CONVOLUTION! *Youtube* [online]. San Bruno, California: Youtube, 2018, 19. března 2018 [cit. 2022-01-09]. Dostupné z: https://www.youtube.com/watch?v=T7o3xvJLuHk

23. TSANG, Sik-Ho. Review: MobileNetV2 — Light Weight Model (Image Classification). *Towards Data Science* [online]. Kanada: Medium, 2019, 19. května 2019 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

24. TSANG, Sik-Ho. Review: MobileNetV1 — Depthwise Separable Convolution (Light Weight Model). *Towards Data Science* [online]. Kanada: Medium, 2018, 14. října 2018 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69

25. ALAKE, Richmond. Understanding and Implementing LeNet-5 CNN Architecture (Deep Learning). *Towards Data Science* [online]. Kanada: Medium, 2020, 25. června 2020 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/understanding-and-implementing-lenet-5-cnn-architecture-deep-learning-a2d531ebc342

26. GUPTA, Prashant. Decision Trees in Machine Learning. *Towards Data Science* [online]. Kanada: Medium, 2017, 17. května 2017 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

27. PATEL, Dhaval. Machine Learning Tutorial Python: 10 - Support Vector Machine. *Youtube* [online]. San Bruno, California: Google, 2018, 19. prosince 2018 [cit. 2022-01-09]. Dostupné z: https://www.youtube.com/watch?v=FB5EdxAGxQg&list=PLeo1K3hjS3uvCeTYTeyfe0-rN5r8zn9rw&index=12

28. REFAELI, David. Sigmoid, Softmax and their derivatives. *The Maverick Meerkat* [online]. Tel Aviv, Izrael: TheMaverickMeerkat.com, 2019, 23. října 2019 [cit. 2022-03-17]. Dostupné z: https://themaverickmeerkat.com/2019-10-23-Softmax/

29. STALLKAMP, J., M. SCHLIPSING, J. SALMEN a C. IGEL. The German Traffic Sign Recognition Benchmark. *Institut für Neuroinformatik* [online]. Ruhr-Universität Bochum, Německo: Institut für Neuroinformatik (INI), 2010, 16. září 2010 [cit. 2022-03-17]. Dostupné z: https://benchmark.ini.rub.de/gtsrb_dataset.html

30. WEI, Jerry. VGG Neural Networks: The Next Step After AlexNet. *Towards Data Science* [online]. Toronto: Towards Data Science, 2019, 3. července 2019 [cit. 2022-03-17]. Dostupné z: https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c

31. CHOLLET, Francois. Transfer learning & fine-tuning. *Keras.io* [online]. Mountain View, California, 2020, 5. května 2020 [cit. 2022-03-17]. Dostupné z: https://keras.io/guides/transfer_learning/

32. Module: tf.keras.applications. *TensorFlow.org* [online]. Mountain View, California, 2022, 11. března 2022 [cit. 2022-03-17]. Dostupné z: https://www.tensorflow.org/api_docs/python/tf/keras/applications

33. STANEK, Mirosław. GTSRB TensorFlow Lite. *GitHub* [online]. Krakow, Polsko: github, 2020, 21. září 2020 [cit. 2022-03-17]. Dostupné z: https://github.com/frogermcs/GTSRB-TensorFlow-Lite

34. Core ML: Integrate machine learning models into your app. *Apple Developer* [online]. Cupertino, California: Apple, 2022 [cit. 2022-03-17]. Dostupné z: https://developer.apple.com/documentation/coreml

35. Apple A15. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022, 12. března 2022 [cit. 2022-03-17]. Dostupné z: https://en.wikipedia.org/wiki/Apple_A15

36. TensorFlow Lite Object Detection iOS Example Application. *GitHub* [online]. San Francisco, California: GitHub, 2021, 6. července 2021 [cit. 2022-03-17]. Dostupné z: https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/ios

37. Tensorflow Lite Core ML delegate. *TensorFlow.org* [online]. Mountain View, California, 2021, 28. ledna 2021 [cit. 2022-03-17]. Dostupné z: https://www.tensorflow.org/lite/performance/coreml_delegate#:~:text=The%20TensorFlow%20Lite%20Core%20ML,0%20and%20latest%20nightly%20releases.

38. AIMOTIVE TEAM. Efficiency, not Utilization or TOPS: why it matters. *Medium.com* [online]. San Francisco, California: Medium, 2021, 18. června 2021 [cit. 2022-03-17]. Dostupné z: https://medium.com/aimotive/efficiency-not-utilization-or-tops-why-it-matters-a4ef1301c5e5

39. https://netron.app/

40. Loss Functions. *ML Glossary* [online]. San Francisco, California: GitHub, 2017 [cit. 2022-03-17]. Dostupné z: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#:~:text=Cross%2Dentropy%20loss%2C%20or%20log,diverges%20from%20the%20actual%20label.

41. BROWNLEE, Jason. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. *Machine Learning Mastery* [online]. San Juan, USA: Machine Learning Mastey, 2021, 13. ledna 2021 [cit. 2022-03-17]. Dostupné z: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=the%20name%20Adam%20is%20derived,Little%20memory%20requirements.

42. KINGMA, Diederik P. a Jimmy BA. Adam: A Method for Stochastic Optimization. *ArXiv* [online]. Cornell University: ArXiv, 2017, 30. ledna 2017 [cit. 2022-03-17]. Dostupné z: https://arxiv.org/abs/1412.6980

43. STANEK, Mirek. Traffic signs classification with retrained MobileNet model. *Think, mobile!* [online]. Krakow, Polsko: Think, mobile!, 2019, 22. ledna 2019 [cit. 2022-03-18]. Dostupné z: https://thinkmobile.dev/mobile-intelligence-traffic-signs-classification-with-retrained-mobilenet-model/

44. Results. *Benchmark.ini.rub.de* [online]. Bochum, Německo: Institut für Neuroinformatik, 2019, 10. května 2019 [cit. 2022-03-18]. Dostupné z: https://benchmark.ini.rub.de/gtsrb_results.html

45. https://paperswithcode.com/sota/traffic-sign-recognition-on-gtsrb

46. WADDINGTON, Joshua. Traffic Sign Classification using Deep Learning. *GitHub* [online]. San Francisco, USA: GitHub, 2017, 18. listopadu 2017 [cit. 2022-03-18]. Dostupné z: https://github.com/joshwadd/Deep-traffic-sign-classification

47. MicronNet: A Highly Compact Deep Convolutional Neural Network Architecture for Real-time Embedded Traffic Sign Classification. Papers with code [online]. San Francisco, USA: Papers with code, 2018, 28. března 2018 [cit. 2022-03-18]. Dostupné z: https://paperswithcode.com/paper/micronnet-a-highly-compact-deep-convolutional

48. LIANG, Frank. Evaluating the Performance of Machine Learning Models. *Towards Data Science* [online]. San Francisco, USA: Medium, 18. dubna 2020 [cit. 2022-03-23]. Dostupné z: https://towardsdatascience.com/classifying-model-outcomes-true-false-positives-negatives-177c1e702810

49. DEEPLIZARD. Fine-Tuning MobileNet On Custom Data Set With TensorFlow's Keras API. *Deeplizard* [online]. Deeplizard, 2020, 28. září 2020 [cit. 2022-03-27]. Dostupné z: https://deeplizard.com/learn/video/Zrt76AIbeh4

50. SRIRAM, G. Fine Tune MobileNet V2. *Kaggle* [online]. Bengaluru, India: Deeplizard, 2020, 2020 [cit. 2022-03-27]. Dostupné z: https://www.kaggle.com/code/sgcuber24/fine-tune-mobilenet-v2/notebook

51. https://keras.io/api/callbacks/

52. Model optimization. *TensorFlow* [online]. Mountain View, USA, 2021, 20. října 2021 [cit. 2022-03-28]. Dostupné z: https://www.tensorflow.org/lite/performance/model_optimization

7.2 Obrazové zdroje

Obrázek 1 (upraveno): BABS, Temi. The Mathematics of Neural Networks. *Medium.com* [online]. Medium, 2017, 14. července 2017 [cit. 2022-01-09]. Dostupné z: https://medium.com/coinmonks/the-mathematics-of-neural-network-60a112dd3e05

Obrázek 2: SHARMA, Sagar. Activation Functions in Neural Networks: Sigmoid, tanh, Softmax, ReLU, Leaky ReLU EXPLAINED !!! *Towards Data Science* [online]. Kanada: Medium, 2017, 6. září 2017 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Obrázek 3: REFAELI, David. Sigmoid, Softmax and their derivatives. *The Maverick Meerkat* [online]. Tel Aviv, Izrael: TheMaverickMeerkat.com, 2019, 23. října 2019 [cit. 2022-03-17]. Dostupné z: https://themaverickmeerkat.com/2019-10-23-Softmax/

Obrázek 4: BROWNLEE, Jason. A Gentle Introduction to the Rectified Linear Unit (ReLU). *Machine Learning Mastery* [online]. San Juan: Machine Learning Mastery, 2019, 9. ledna 2019 [cit. 2022-01-09]. Dostupné z: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Obrázek 5 a 6: SHARMA, Sagar. What the Hell is Perceptron?: The Fundamentals of Neural Networks. *Towards Data Science* [online]. Kanada: Medium, 2017, 9. září 2017 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

Obrázek 7 (upraveno): IBM CLOUD EDUCATION. Neural Networks. *IBM.com* [online]. Armonk, New York State: IBM.com, 2020, 17. srpna 2020 [cit. 2022-01-08]. Dostupné z: https://www.ibm.com/cloud/learn/neural-networks

Obrázek 8 a 12: SAHA, Sumit. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. *Towards Data Science* [online]. Kanada: Medium, 2018, 15. prosince 2018 [cit. 2022-01-08]. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Obrázek 9 (upraveno): IBM CLOUD EDUCATION. Convolutional Neural Networks. *IBM.com* [online]. Armonk, New York State: IBM.com, 2020, 20. října 2020 [cit. 2022-01-08]. Dostupné z: https://www.ibm.com/cloud/learn/convolutional-neural-networks

Obrázek 10 a 11: VUDMOULIN a FVISION. Convolution arithmetic. *GitHub.com* [online]. San Francisco: GitHub, 2019, 12. dubna 2019 [cit. 2022-01-08]. Dostupné z: https://github.com/vdumoulin/conv_arithmetic

Obrázek 13: LIHUA, Wen a Kang-Hyun JO. Traffic sign recognition and classification with modified residual networks. ResearchGate [online]. Jižní Korea, Prosinec 2017 [cit. 2022-03-23]. Dostupné z: https://www.researchgate.net/figure/The-total-43-classes-in-GTSRB-From-top-to-bottom-there-are-four-categories_fig1_322945549

Obrázek 24: OSOUCH, Marek. Na dvoukilometrovém průtahu Blanskem řidiče zpomalí pět kruhových objezdů. *Idnes.cz* [online]. Česká republika, 2021, 7. června 2021 [cit. 2022-03-18]. Dostupné z: https://www.idnes.cz/brno/zpravy/kruh-objezd-stavba-blansko-pomala-cesta-doprava-prutah-pet.A210604_611373_brno-zpravy_mos1

Obrázek 25: ULRYCHOVÁ, Bára. Přehled a význam zákazových dopravních značek. *Portál řidiče* [online]. Česká republika, 2022, 25. 1. 2022 [cit. 2022-03-18]. Dostupné z: https://www.portalridice.cz/clanek/zakazove-dopravni-znacky-druhy-a-jejich-vyznam

Obrázek 26: DUŠEK, Martik. Přehlednější doprava, víc parkovacích míst. Most zruší desítky dopravních značek. *Český rozhlas Sever* [online]. Česká republika: Český rozhlas, 2017, 1. únor 2017 [cit. 2022-03-18]. Dostupné z: https://sever.rozhlas.cz/prehlednejsi-doprava-vic-parkovacich-mist-most-zrusi-desitky-dopravnich-znacek-6841852

7.3 Softwarové zdroje

1. Visual Studio Code: MICROSOFT. *Visual Studio Code* [software]. In: . 3. března 2022 [cit. 2022-03-23]. Dostupné z: https://code.visualstudio.com/

2. Adobe Illustrator: ADOBE. *Illustrator* [software]. In: . 26. října 2021 [cit. 2022-03-23]. Dostupné z: https://www.adobe.com/products/illustrator.html

3. Xcode: APPLE. Xcode. Apple Developer [software]. 14. března 2022 [cit. 2022-03-23]. Dostupné z: https://developer.apple.com/xcode/

4. Google Colab Pro. *Google Colab* [software]. Mountain View, USA, 2021 [cit. 2022-03-28]. Dostupné z: https://colab.research.google.com/?utm_source=scs-index

Příloha 1 Tabulka číselného pořadí a názvů dopravních značek sady GTSRB

Číslo třídy	Autorský název	Český normovaný název
0	20_speed	Nejvyšší dovolená rychlost 20 km/h
1	30_speed	Nejvyšší dovolená rychlost 30 km/h
2	50_speed	Nejvyšší dovolená rychlost 50 km/h
3	60_speed	Nejvyšší dovolená rychlost 60 km/h
4	70_speed	Nejvyšší dovolená rychlost 70 km/h
5	80_speed	Nejvyšší dovolená rychlost 80 km/h
6	80_lifted	Konec nejvyšší dovolené rychlosti
7	100_speed	Nejvyšší dovolená rychlost 100 km/h
8	120_speed	Nejvyšší dovolená rychlost 120 km/h
9	no_overtaking_general	Zákaz předjíždění
10	no_overtaking_trucks	Zákaz předjíždění pro nákladní automobily
11	right_of_way_crossing	Křižovatka s vedlejší pozemní komunikací
12	right_of_way_general	Hlavní pozemní komunikace
13	give_way	Dej přednost v jízdě!
14	stop	Stůj, dej přednost v jízdě!
15	no_way_general	Zákaz vjezdu všech vozidel (v obou směrech)
16	no_way_trucks	Zákaz vjezdu nákladních automobilů
17	no_way_one_way	Zákaz vjezdu všech vozidel
18	attention_general	Jiné nebezpečí
19	attention_left_turn	Zatáčka vlevo
20	attention_right_turn	Zatáčka vpravo
21	attention_curvy	Dvojitá zatáčka, první vlevo
22	attention_bumpers	Nerovnost vozovky
23	attention_slippery	Nebezpečí smyku
24	attention_bottleneck	Zúžená vozovka z jedné strany
25	attention_construction	Práce na silnici
26	attention_traffic_light	Světelné signály
27	attention_pedestrian	Chodci
28	attention_children	Děti
29	attention_bikes	Cyklisté
30	attention_snowflake	Náledí
31	attention_deer	Zvěř
32	lifted_general	Konec všech zákazů
33	turn_right	Přikázaný směr jízdy vpravo
34	turn_left	Přikázaný směr jízdy vlevo
35	turn_straight	Přikázaný směr jízdy rovně
36	turn_straight_right	Přikázaný směr jízdy rovně a vpravo
37	turn_straight_left	Přikázaný směr jízdy rovně a vlevo
38	turn_right_down	Přikázaný směr objíždění vpravo
39	turn_left_down	Přikázaný směr objíždění vlevo
40	turn_circle	Kruhový objezd
41	lifted_no_overtaking_general	Konec zákazu předjíždění
42	lifted_no_overtaking_trucks	Konec zákazu předjíždění pro nákladní automobily

Příloha 2 Grafické znázornění architektury sítě z aplikace Netron