VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Vojtěch Mikulec

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

DETEKCE POHYBUJÍCÍCH SE OBJEKTŮ VE VIDEU S VYUŽITÍM NEURONOVÝCH SÍTÍ POMOCÍ ANDROID APLIKACE
OBJECT DETECTION IN VIDEO USING NEURAL NETWORKS AND ANDROID APPLICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Vojtěch Mikulec

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Vojtěch Myška

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Vojtěch Mikulec ***ID:*** 197774

Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Detekce pohybujících se objektů ve videu s využitím neuronových sítí pomocí Android aplikace

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s technikami konvolučních neuronových sítí, problematikou detekce objektů v obraze a zpracujte aktuální stav vědy a techniky v této oblasti. Na základě získaných poznatků navrhněte funkční řešení problematiky klasifikace účastníků dopravního provozu pomocí mobilních zařízení s operačním systémem Android. Za účelem dosažení stanoveného cíle vytvořte Android aplikaci, která bude v reálném čase pomocí neuronových sítí klasifikovat účastníky dopravního provozu a ukládat časové značky klasifikace. Pro nasazení algoritmů neuronových sítí lze využít například nástroj TensorFlow Lite. Za účelem kategorizace lze použít i některé předtrénované modely, které budou následně dotrénovány na vlastně vytvořené datové množině. Dosažené výsledky vhodně srovnejte a diskutujte.

DOPORUČENÁ LITERATURA:

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimalspeed and accuracy of object detection,” 2020.

[2] Developer Guides. Developer Guides [online]. Dostupné z: https://developer.android.com/guide

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Vojtěch Myška

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato diplomová práce se zabývá zrealizováním funkčního řešení problematiky klasifikace účastníků dopravního provozu pomocí mobilních zařízení s operačním systémem Android. Cílem je vytvořit Android aplikaci, která klasifikuje vozidla v reálném čase za použití zadní kamery a ukládá časové značky klasifikace. Testování probíhá převážně na vlastní, různě modifikované datové množině. Je natrénováno celkem pět modelů a změřeno zatížení hardwaru při použití každého z nich. Nejlepší přesnosti klasifikace dosahuje předtrénovaný model sítě MobileNet, který je dotrénován o 6 tříd – 62,33 %. Výsledky jsou shrnuty a v závěru je formulováno, jakým způsobem je možné rychleji a přesněji analyzovat dopravní data.

KLÍČOVÁ SLOVA

Android, aplikace, klasifikace obrazu, konvoluční neuronová síť, přenosové učení, vozidla, zpracování dat

ABSTRACT

This master's thesis deals with the implementation of functional solution for classifying road users using mobile device with Android operating system. The goal is to create Android application which classifies vehicles in real time using rear-facing camera and saves timestamps of classification. Testing is performed mostly with own, diversely modificated dataset. Five models are trained and their performance is measured in dependence on hardware. The best classification performance is from pretrained MobileNet model where transfer learning with 6 classes of own dataset is used – 62,33 %. The results are summarized and a method for faster and more accurate traffic analysis is proposed.

KEYWORDS

Android, application, image classification, convolutional neural network, transfer learning, vehicles, image processing

MIKULEC, Vojtěch. *Detekce pohybujících se objektů ve videu s využitím neuronových sítí pomocí Android aplikace*. Brno, 2021, 86 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Vojtěch Myška

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Detekce pohybujících se objektů ve videu s využitím neuronových sítí pomocí Android aplikace“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora

PODĚKOVÁNÍ

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

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Se zpracováním obrazu v jakékoliv formě, ať už statické (fotografie) nebo dynamické (video), přicházíme do kontaktu stále častěji. Za poslední dekádu se oblast informačních technologií prudce rozrostla, což se projevilo i do odvětví klasifikace obrazu. Využívat technik zpracování obrazu tak může každý, kdo má k dispozici dnes už hojně dostupný hardware pro spouštění výpočtů. Tak vznikají aplikace, které posouvají toto odvětví rychleji dopředu. Existuje mnoho metod, které dokáží naučit stroj, aby se sám „rozhodoval“ podle předem daných instrukcí a pravidel. Jednou z těchto metod je využití umělých neuronových sítí, které po natrénování dokáží predikovat výsledek. Jejich správná predikce je ale závislá na předložených datech. Rychlost klasifikace pak závisí na použitém hardwaru. Při implementaci těchto natrénovaných algoritmů do mobilních zařízení může vznikat problém s nedostatkem jeho výkonu a tím pomalé nebo nedostatečné klasifikace.

Úkolem této práce je vytvoření mobilní aplikace a použití neuronové sítě, která bude správně klasifikovat objekty pohybující se v reálném čase s využitím kamery mobilního telefonu. Zároveň budou v rámci aplikace ukládány časové značky klasifikace. Pro tento úkol je nejprve třeba mít k dispozici vhodnou datovou množinu a následně natrénovat neuronovou síť tak, aby bylo dosaženo přijatelné přesnosti klasifikace. Na konec bude třeba model převést a implementovat do vytvořené mobilní aplikace. Při následné klasifikaci bude sledována přesnost predikce, využití výkonu mobilního telefonu a rychlost klasifikace neuronové sítě.

V praktické části diplomové práce jsou natrénovány modely konvolučních neuronových sítí a je navrhnuta a vytvořena Android aplikace tak, aby dokázala využít tento model ke klasifikaci dopravních prostředků. Důraz je v této práci kladen především na schopnost klasifikace a využití hardwaru.

Hlavním přínosem práce je otestování proveditelnosti náročných výpočtů na podstatně omezeném hardwaru mobilního telefonu, který musí zvládat zpracovávat větší množství instrukcí v reálném čase. Vzhledem k tomu, že detekce je orientována na dopravní prostředky, je možné se díky ní dostat ke kvantitativním výsledkům zatížení dopravní sítě nebo automatizace analýzy dopravy. V neposlední řadě bude vytvořena vlastní datová množina a tím bude docíleno zjemnění již existujících množin na více kategorií pro přesnější klasifikaci.

Práce je strukturována následovně. V první části se pojednává o teorii strojového učení a využití neuronových sítí pro zpracování obrazu. Druhá část popisuje současný stav vědy a techniky tohoto odvětví a popisuje konvoluční neuronové sítě. Třetí část je experimentální, obsahuje návrh aplikace a popisuje využité nástroje k natrénování modelů. Poslední část se věnuje výsledkům natrénovaných modelů. Výsledky jsou nakonec zhodnoceny a shrnuty v závěru.

1 Teoretický úvod

Pro vývoj aplikací zpracování obrazu je využíváno mnoha různých metod, které si kladou za cíl celý proces klasifikace co nejvíce zautomatizovat a zároveň dosahovat co nejpřesnějších výsledků. Při klasifikaci obrazových dat je obvykle využíváno strojového učení, kdy jsou stroji předložena obrazová data, která jsou zpracována na iteračním principu pomocí neuronových sítí. V závěru tohoto procesu je každý vzorek z dat přiřazen do kategorie, která je s danou přesností správná a s tímto výsledkem je pak dále možné pracovat. Například vyladit algoritmus k dosažení vyšší přesnosti nebo v případě spokojenosti uživatele nasazení konečného modelu do praxe, kde může vykonávat natrénovanou automatizační činnost. Pro zahájení a celý proces trénování je důležité mít k dispozici dostatek kvalitních dat. V opačném případě bývají natrénované modely nepřesné a v praxi jsou poté nepoužitelné [1]. Pro trénování je také zapotřebí dostatečný výpočetní výkon, jelikož je s předloženými daty prováděno mnoho matematických operací ve více vláknech hardwaru [2]. Výsledný model je poté dostupný k nahrání do zařízení, která snímají obraz a s využitím modelu v reálném čase určují, co se na obraze právě nachází. Výstup lze použít za účelem následného analytického zpracování dat.

Tato práce se zabývá použitím vhodné neuronové sítě a datové množiny k tomu, aby byla v reálném čase co nejpřesněji klasifikována pohybující se vozidla, konkrétně jejich typ. Výsledný model bude překonvertován do formátu kompatibilního s operačním systémem Android a bude využit ve vytvořené mobilní aplikaci pro klasifikaci vozidel.

1.1 Strojové učení

Strojové učení je jedním z oborů umělé inteligence, jehož cílem je vytvoření reprezentativního a zjednodušeného modelu z velkého objemu dat. Použití strojového učení je možné v jakémkoliv oboru, kde existuje dostatek dat ke zpracování. Například v bankovnictví je potřeba analyzovat data pro vytvoření různých aplikací pro stávající i budoucí zákazníky, detekovat podvodné jednání nebo předpovídat vývoj burzy. V medicíně se strojové učení začíná uplatňovat stále častěji při diagnostikování pacientů s různými chorobami, například s roztroušenou sklerózou nebo Parkinsonovou chorobou [3]. V telekomunikacích jsou analyzována schémata hovorů pro optimalizaci celé telekomunikační sítě a zkvalitnění služeb a ve vědě může být stroji předloženo obrovské množství dat z astronomie nebo biologie, které je možné počítačem poměrně rychle zpracovat. Výše vypsaná uplatnění jsou pouze jedny z mnoha dalších, ve kterých mohou být algoritmy strojového učení nasazeny.

Strojové učení ale není pouze souhrn algoritmů, který je schopen zpracovávat databáze, ale je to součást oboru umělé inteligence. Aby byl systém považován za inteligentní, musí být schopen se učit a přizpůsobovat se změně prostředí. Pojem „strojové učení“ je možné vysvětlit jako programování počítače k optimalizaci libovolného kritéria po předložení dat z minulosti. Výsledný model využívá teorii statistiky pro vytváření matematických modelů a může tak plnit popisnou funkci pro získání vybraných informací z dat, prediktabilní pro určení předpovědi budoucího vývoje trendu z minulosti nebo obojí zároveň. V první fázi učení probíhá trénování na datech, kdy je vytvořen optimalizovaný model problému. V druhé fázi je zapotřebí, aby vytvořená reprezentace a algoritmická řešení byla efektivní a schopna klasifikace [4].

1.1.1 Proces strojového učení

V rámci strojového učení jsou rozlišovány tři metody učení (trénování):

1. Učení s učitelem *(supervised learning)*
2. Učení bez učitele *(unsupervised learning)*
3. Posílené učení *(reinforcement learning)*

Tato práce se zabývá problematikou neuronových sítí, jejichž výsledkem je klasifikační model. Aplikace, které ve své konečné fázi dokáží klasifikovat nebo zpracovávat data k tomu musí být nejprve naučeny. Pro tuto potřebu je nutné mít k dispozici roztříděná data, která jsou na výstupu neuronové sítě přiřazena do jedné ze tříd. Tento způsob trénování se označuje jako tzv. učení s učitelem, kdy vývojář aplikace určuje vstupní data a následně po zpracování neuronovou sítí k nim přiřadí korespondující data. Uživatel je tak obeznámen s tím, jestli zvolil vhodná data nebo parametry sítě a jestli je model správně natrénovaný. Celý proces trénování je tak v režii uživatele a parametry mohou být mezi jednotlivými kroky upravovány [5].

1.2 Neuronové sítě

První zmínka o neuronových sítích (živých struktur) pochází z roku 1943, kdy byl publikován článek neuropsychologa W. McCullocha a neurovědce W. Pittse. Přibližná funkce mozku byla již delší dobu známa, článek [8] ale pojednává o podrobnější funkci neuronů a popisuje sestavení fyzického modelu pomocí elektrických obvodů. K podobě strojových neuronových sítí, jak jsou známé dnes, vede desítky let dlouhá cesta. První model jednoduché neuronové sítě byl sestaven v roce 1958 F. Rosenblattem, který obsahuje jednu vrstvu s neuronem, provádějící matematické operace k určení vstupu do jedné ze dvou tříd [10].

Název „neuronová síť“ pochází z neurologických struktur v mozku, které jsou

tvořeny živými neurony. Mezi těmito neurony probíhá neustálá komunikace elektrickými impulzy přes neuronové výběžky, které se nazývají dendrity. Tyto výběžky jsou propojeny s dalšími neurony. Uvádí se, že každý neuron v mozku může být spojen s 1000 – 5000 dalšími neurony [7]. Uvedeným způsobem se vzruchy v mozku šíří skrz rozsáhlou síť neuronů, které od předchozích přijímají signály přes synapse, což je termín pro místo spojení dvou neuronů. Následně proces vyústí v učení organismu, kdy se opakováním dané činnosti spoj mezi neurony posiluje – silnější spoj znamená lépe naučenou činnost. Poslední fází, která následuje po učení je pamatování si. To, že je možné si činnost zapamatovat a podruhé ji vykonat stejně, je v mozku docíleno stimulací zakončení výběžků neuronu (tzv. axonu). Zakončení axonů se tak stává citlivější na vzruchy a výsledkem je zapamatování si dané činnosti.

Ve strojových strukturách neuronů je možné najít mnoho podobností s těmi živými. Kromě názvu, kdy biologie i strojové učení používá výraz „neuron“, se v oblasti strojového učení používá i výraz „perceptron“. Ten lze připodobnit k neuronu v živých strukturách stejně tak jako „axony“, které v neuronových sítích zastupují „váhy“. Všechny tyto neurony jsou spolu propojené a tvoří tak síť, stejně jako v mozku. Jediná odlišnost je tak v jejich „programovatelnosti“, kdy živé neurony jsou naprogramovány a posíleny přes libovolnou činnost mozku. Strojové neurony jsou pak souhrnem matematických pravidel, které musí být splněny pro propuštění vstupní informace v neuronu do dalších neuronů, které navazují [9].

Neurony a jejich seskupení (neuronové sítě) tak mohou vykonávat dva druhy činností:

1. Učení – Neuron se iteračně učí rozpoznávat příznaky. V případě obrazových dat se může jednat o malá seskupení pixelů, která k sobě patří. Model neuronové sítě je pak celé seskupení neuronů a jejich propojení – vah mezi neurony.
2. Trénování – Předání natrénovaných příznaků na výstup neuronu přes váhy [9].

V době vysoké konkurence technologických firem na trhu se s neuronovými sítěmi má možnost setkat kdokoliv z nás. Stačí, aby si člověk koupil chytrý mobilní telefon s fotoaparátem a spolu s ním dostane i jakýmkoliv způsobem natrénované a implementované modely neuronových sítí. Některé technologické společnosti již začaly vyhrazovat celá jádra grafických i výpočetních procesorů pouze pro výpočty algoritmů zpracování obrazu, z čehož lze předpokládat jejich zaměření na automatizaci a následnou personalizaci obsahu. Například společnost Apple, která se mimo jiné zabývá vývojem mobilních telefonů, uvedla na trh v roce 2017 mobilní čip s označením „*Apple A11 Bionic*“. Tento čip zahrnuje i část, která se nazývá „*Neural Engine*“ a právě tato část čipu má za úkol pomocí neuronových sítí zpracovávat obraz. Podle oficiálních informací je schopna zpracovat až 600 miliard operací za vteřinu a stará se tak o bezpečné rozpoznávání tváře při odemykání mobilního telefonu *(Face ID)*, personalizované animované emotikony *(Animoji)* a další úkoly, které mohou být efek-

tivně vyřešeny pouze za použití technik strojového učení [6]. To je možné vnímat jako důkaz rozvoje strojového učení v mobilních zařízeních. Dalšími příklady využití zpracování obrazu v mobilních telefonech může být automatické rozpoznání obličejů na pořízené fotografii.

1.2.1 Optimalizační algoritmy

Jak bylo uvedeno výše, neurony, které spolu tvoří sítě, se mohou učit a trénovat. Při učení dochází k jejich optimalizaci, kdy pro předání výsledku do dalšího neuronu je nutné najít správnou hodnotu funkce. Před začátkem učení je třeba definovat model a úkol, který má vykonat. Tento model se skládá z parametrů a architektury. Parametry určují, jak přesné výsledky bude model podávat pro danou architekturu sítě. Je ale pouze na vývojáři, jakým způsobem se dopracuje k přijatelným hodnotám parametrů a tím i klasifikace. Ve velké většině případů je tak zapotřebí definovat tzv. ztrátovou funkci *(loss function)*, která iteračně zhodnocuje, jak je model úspěšný. Cílem je najít vhodné parametry a tím minimalizovat tuto funkci [11].

Pro tento účel se používá tzv. gradient. Gradient je diferenciální operátor a zastupuje směrnici přímky (vektorová funkce), která je obecnou derivací funkce o více proměnných. Vzhledem k tomu, že vektory gradientu směřují podle nejvyšší frekvence nárůstu funkce, tak by se dalo říct, že gradient znamená „směr růstu“. Gradient je invariantní veličina, která nezávisí na soustavě souřadnic. V kartézské soustavě je ale pro funkci $w = f(x, y, z)$ definován vztahem

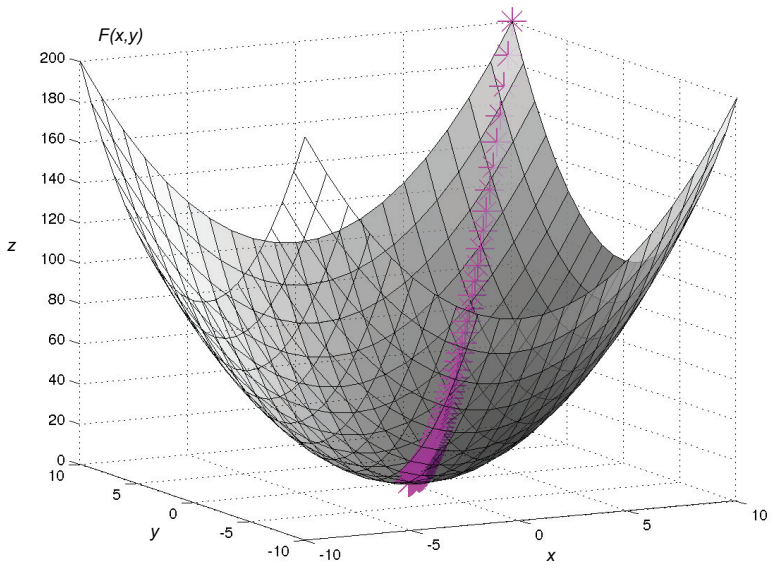
$$\nabla f = \mathrm{grad} f = \frac{\partial f}{\partial x}(x, y, z) + \frac{\partial f}{\partial y}(x, y, z) + \frac{\partial f}{\partial z}(x, y, z), \tag{1.1}$$

kde x, y, z jsou jednotlivé osy soustavy souřadnic [14].

Optimalizační algoritmy tak staví na principu postupné iterace směrem ke globálnímu minimu funkce. Níže jsou popsány optimalizační algoritmy, které se využívají pro optimalizaci v neuronových sítích, a se kterými bude experimentováno v praktické části této práce.

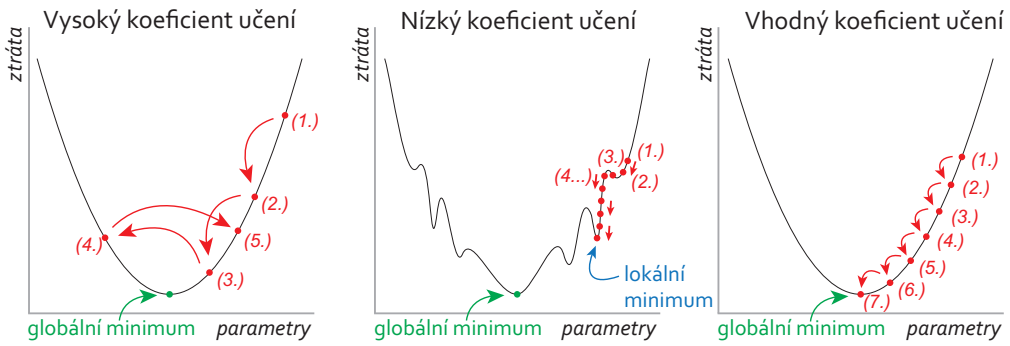
1. **Gradientní sestupování**
   Gradientní sestupování je způsob, kterým lze minimalizovat konvexní funkci $F(x, y)$ závislou na parametrech modelu, kde $x \in \mathbb{R}$. Jak lze vidět na obrázku 1.1, který znázorňuje trojrozměrnou vizualizaci funkce $F = z = x^2 + y^2$, nastavením iteračního kroku je potřebné najít globální minimum funkce a tím ji optimalizovat. Kategorie gradientního sestupu zahrnuje další dva algoritmy, které jsou ve svém jádru podobné, ale liší se počáteční inicializací a průběhem. Jejich podstata – hledání globálního minima funkce je ale společná. Těmito

algoritmy jsou náhodné (stochastické) gradientní sestupování (SGD) a dávkové gradientní sestupování (BGD). Z vizualizace je zřejmé, že inicializace optimalizace probíhá v lokálním maximu funkce. Tento krok je většinou určován experimentálně a nemusí tak začínat vždy na maximu funkce. Důležité pro trénování je, aby vždy dospěl do globálního minima funkce a tím dosáhl co největší optimalizace [12]. Cesta k tomuto minimu není ale vždy tak jednoduchá, jako v případě vizualizované funkce na obrázku 1.1. Funkce mohou mít různý tvar a kromě globálních minim a maxim mohou obsahovat i lokální. Při trénování neuronových sítí je důležité správně nastavit tzv. koeficient učení *(learning rate)*, který definuje rychlost změny právě optimalizovaného gradientu po jednotlivých krocích. Jeho nastavení bývá experimentální, kdy je v počátku zvolena hodnota, která je dále se získanými výsledky upravována. Zvolení příliš vysokého koeficientu učení způsobí přeskakování mezi stěnami funkce a ve výsledku nebude globálního minima dosaženo. Pokud je naopak zvolen příliš malý koeficient učení, může být dosaženo výsledku, ale optimalizace není důsledná, protože se může jednat pouze o lokální minimum a do globálního se nedostane, jak je znázorněno na obrázku 1.2. Uživatel může v tomto případě použít funkci `momentum`, která pomůže gradientu z lokálního minima a učení může pokračovat dále správným směrem ke globálnímu minimu.



Obr. 1.1: Vizualizace gradientního sestupování funkce $F(x, y)$ [13].

2. **Stochastické gradientní sestupování (SGD)**
   Náhodné gradientní sestupování patří mezi nejpoužívanější optimalizační algoritmy používané v neuronových sítí. Jeho cílem je minimalizovat empirické



Obr. 1.2: Případy pro různé hodnoty koeficientu učení.

riziko modelu opakovaným počítáním gradientu ztrátové funkce na jednom trénovacím vzorku nebo malé dávce vzorků. Podle výpočtu pak vhodně upravuje parametry sítě. Metoda, využívající SGD je škálovatelná, robustní a podává dobré výsledky v širokém spektru problémů neuronových sítí (silně konvexní i komplexní nekonvexní problémy). Článek [15] uvádí shrnutí, že každý model, natrénovaný metodou SGD v odůvodněném časovém úseku, vykazuje malou chybu generalizace. Chybou generalizace se myslí očekávaná chyba na vzorcích datové množiny, které nebyly neuronovou sítí „viděny“ [16].

SGD je tak možné vnímat jako iterativní metodu pro optimalizaci funkce s vhodnými parametry jakým je například jemnost (rozlišitelnost maxim a minim). Jinými slovy je to také náhodná aproximace výše popsaného gradientního sestupování, kdy SGD nahrazuje celý gradient, který je vypočítaný z celé datové množiny. Nahrazen je přibližným odhadem, vypočítaného z jednoho vzorku nebo z náhodně vybrané menší podmnožiny dat. Výsledkem tak mohou být rychlejší iterace ovšem výměnou za nižší rychlost konvergence [17].

3. **Propagace efektivní hodnoty (RMSprop)**
   RMSprop je optimalizátor, který využívá velikosti posledních gradientů k normalizování dalších. V rámci RMS *(root mean square)* je třeba zachovat pohyblivý průměr gradientů, kterým se poté vydělí aktuální gradient. Stejně jako u předchozích optimalizátorů lze přidat funkci `momentum` nebo upravovat rychlost kroku. Jakmile oba tyto parametry míří do stejného bodu funkce, rychlost kroku je vynásobena členem $(1 + \texttt{prizpusobeny\_krok})$, v ostatních případech je vynásobena $(1 - \texttt{prizpusobeny\_krok})$. Konstanta `prizpusobeny_krok` vyjadřuje přizpůsobení rychlosti kroku. Minimum a maximum rychlosti kroku jsou tak brány v úvahu a příliš vysoké hodnoty jsou vyfiltrovány podle potřeby. Tento optimalizátor má několik výhod mezi které patří například jeho robustnost a schopnost vypořádat se s náhodnými úkoly, což je vhodné pro učení neuronových sítí po malých dávkách vzorků [18].

4. **Adaptivní odhad momentu (Adam)**
   Optimalizátor Adam je algoritmus pro optimalizace stochastických funkcí založen na adaptivních odhadech. Metoda využívající tento optimalizátor adaptivně počítá rychlost kroků pro různé parametry získané z odhadů při prvním a druhém kroku gradientu. Jeho výhodami jsou například nízká náročnost na paměť, rozsah aktualizace parametru je invariantní vůči změně rozměrů gradientu, nebo že nevyžaduje nasazení na výhradně statické úkoly [19].

1.2.2 Proces aktualizace neuronů

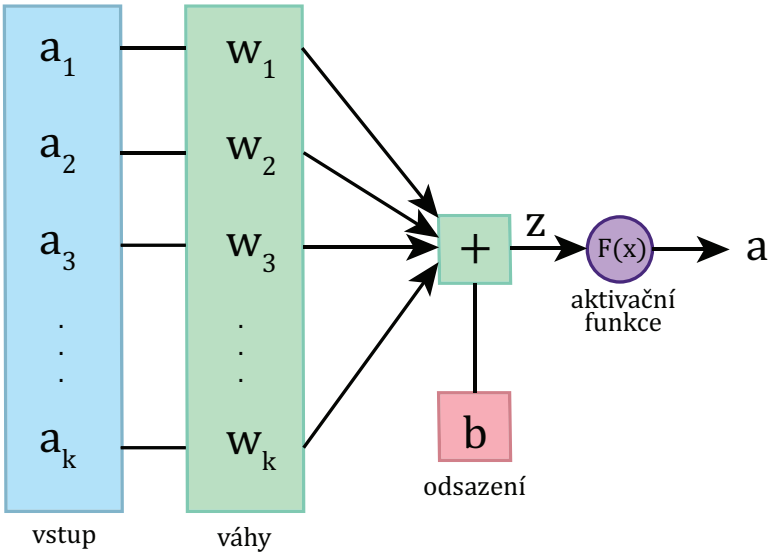
Na vysvětlení gradientních algoritmů pro optimalizaci neuronových sítí navazují další procesy, které jsou nezbytné pro aktualizaci perceptronů a tím způsobenou propagaci dat. Proces učení neuronových sítí je možné považovat za souhrn matematických operací nad vzorky dat, kdy se nejprve vstup do neuronové sítě vynásobí váhami (propojení mezi perceptrony) a dále přičte tzv. odsazení *(bias)*.
Váhy jsou parametrem neuronových sítí, které ve skrytých vrstvách transformují vstupní data. Jakmile data vstoupí do sítě, je provedeno vynásobení váhami a výsledek je buď zaznamenán nebo propuštěn do další vrstvy sítě. Váhy i odsazení jsou parametry, které si síť po dobu učení upravuje. Před samotným započetím učení jsou tyto parametry nastaveny náhodně. V průběhu trénování jsou upravovány tak, aby poskytovaly správný výstup. Tyto dva parametry se liší v rozsahu jejich vlivu na učení. Odsazení reprezentuje, jak daleko jsou předpovědi od zamýšleného výstupu. Nízká hodnota odsazení značí, že neuronová síť více předpokládá formu výstupu. Naopak vysoká hodnota značí, že správná forma výstupu je sítí méně předpokládána. Váhy je možné si představit jako sílu propojení mezi neurony, které ovlivňují velikost vlivu, který může mít změna vstupních dat na výstupní. Nízká hodnota vah nijak nezmění výstup, naopak čím vyšší hodnota vah, tím větší bude změna výstupu z neuronu [20]. Aktualizaci vah z lze matematicky vyjádřit jako

$$z = a_1 w_1 + a_2 w_2 + a_3 w_3 + a_k w_k + b. \tag{1.2}$$

1.2.3 Aktivační funkce

Každý neuron v síti má svá pravidla propouštění dat, která se v angličtině označují jako *„firing rules“*. Tato pravidla jsou popsána aktivačními funkcemi a jejich úkolem je modulace výstupního signálu neuronu, aby mohl být v následující vrstvě přijat. Aktivační funkce tak zajišťují postup příznaků z jedné vrstvy do následující pomocí transformace skalárního vstupu na skalární výstup.

Funkci, která zajišťuje aktivaci omezené amplitudy a rozsahu výstupu neuronu se



Obr. 1.3: Funkce neuronů.

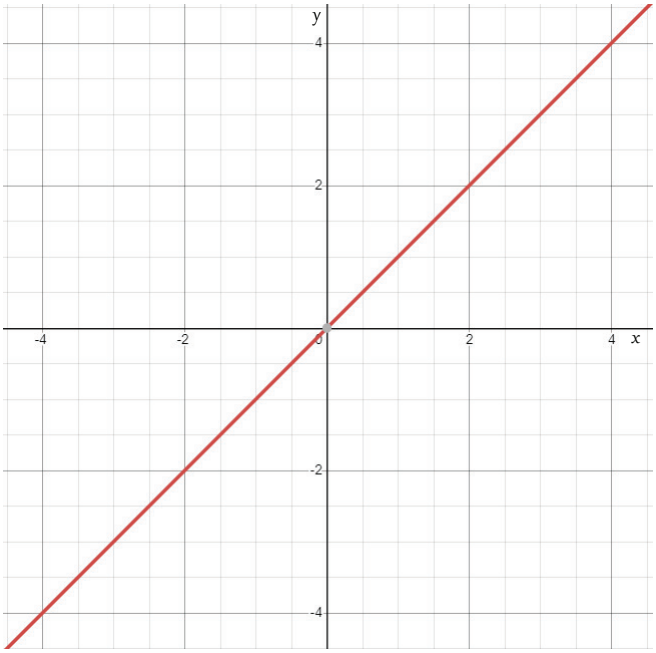
také zjednodušeně říká „zplošťovací funkce“. Tato funkce zploští amplitudu výstupního signálu na konečnou hodnotu. Základních aktivačních funkcí je 10 a podle parametrů sítě a přesnosti výsledné klasifikace lze síť experimentálně upravovat dokud není nalezena vyhovující konfigurace. Níže jsou uvedeny a popsány nejpoužívanější aktivační funkce, které jsou použity v experimentální části této práce [21].

1. **Lineární aktivační funkce**
   Tato funkce je přímo proporcionální vstupu neuronu. Je popsána vztahem

$$F(x) = ax, \tag{1.3}$$

   kdy hodnota proměnné a může být jakákoliv konstanta vybraná uživatelem. Na vyobrazené funkci lze vidět, že derivát funkce $F(x)$ není roven nule, ale hodnotě použité konstanty, v tomto případě 1. Gradient tak není nulový, ale má danou hodnotu, která je nezávislá na vstupní hodnotě. Použití lineární aktivační funkce nemá moc výhod, protože při jejím použití nemá neuronová síť snahu napravovat své chyby, když při každé iteraci má gradient stejnou hodnotu. Síť ani nebude schopna identifikovat komplexní vzorce z dat, je tak vhodná pro velmi jednoduché úkoly [21].
2. **Sigmoid**
   Funkce Sigmoid je jedna z nejvíce používaných funkcí v neuronových sítích z důvodu její nelinearity. Transformuje hodnoty reálných čísel do intervalu $\langle 0, 1\rangle$. Funkce je popsána vztahem

$$F(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{1.4}$$



Obr. 1.4: Graf lineární aktivační funkce pro $a = 1$.

a její graf je na obrázku 1.5. I přes její popularitu a četnost využití je ale v dnešní době čím dál častěji nahrazována jinými funkcemi z důvodu přecházení do saturace. Funkce má po derivaci prvního řádu tvar

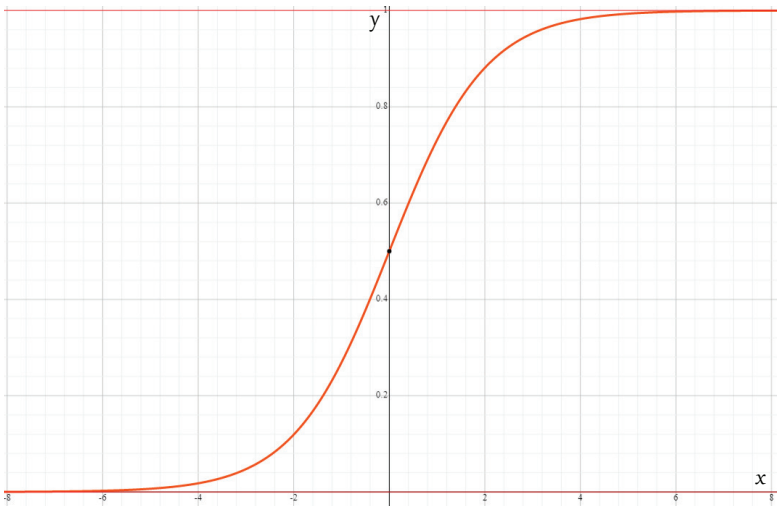
$$F'(x) = \frac{\mathrm{e}^{-x}}{(1 + \mathrm{e}^{-x})^2} \tag{1.5}$$

a po dosazení konstanty $x = 0$ je dosaženo výsledku $0,25$. To znamená, že čím bude v sítí funkce četnější, tím více chyb způsobí při trénování (a to až $4\times$ více chyb v každé následující vrstvě neuronové sítě). Je vhodná pro použití na problémy binární klasifikace, kdy se funkce umísťuje do poslední výstupní vrstvy. Není ale vhodná pro inicializaci neuronové sítě z malých a náhodných vah právě kvůli zmíněnému šíření chyb. Z těchto důvodů se často nahrazuje jinými aktivačními funkcemi hyperbolické povahy (například funkce *Tanh*, která má na rozdíl od Sigmoidy střed v bodě 0) [21, 22].

3. **Tanh**
   Funkce Tanh (hyperbolická tangenta), je velmi podobná funkci Sigmoid, ale jak bylo zmíněno, je symetrická kolem nuly, jemnější a její rozsah zahrnuje čísla od $\langle -1, 1 \rangle$. Její výstup je dán

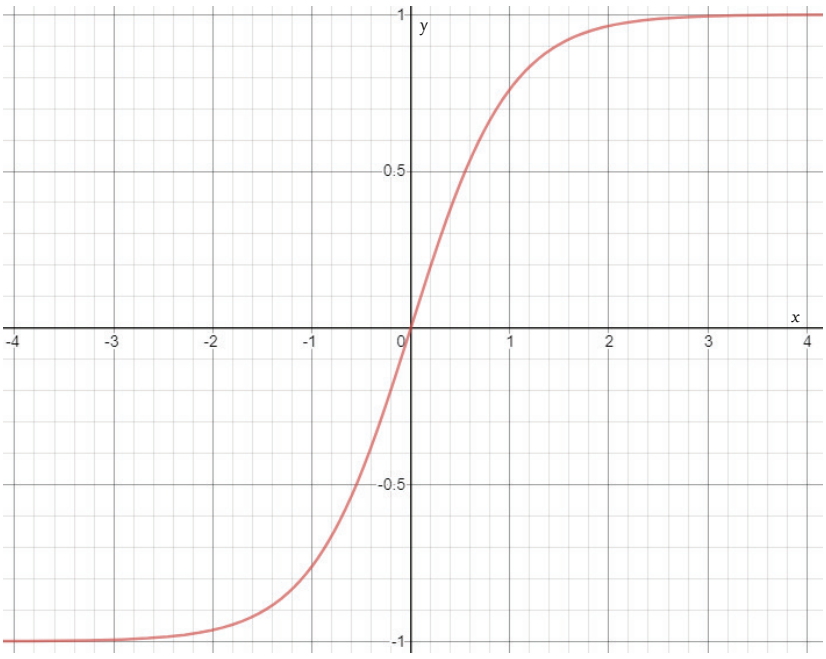
$$F(x) = \left(\frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}} \right). \tag{1.6}$$

   Funkce Tanh se stává preferovanější funkcí, protože ve vícevrstvých sítích podává lepší výsledky při trénování než funkce Sigmoid. Nicméně neodstraňuje



Obr. 1.5: Graf funkce Sigmoid.

její nedostatek mizejícího gradientu. Při trénování s implementovanou funkcí Tanh také vzniká problém neaktivních neuronů, který řeší funkce ReLU. Její hlavní výhodou tak zůstává její symetričnost kolem nuly, kdy tato vlastnost výrazně pomáhá procesu zpětné propagace. Funkce Tanh je používána především v rekurentních neuronových sítích pro zpracovávání řeči (například překlad jazyků) [22].



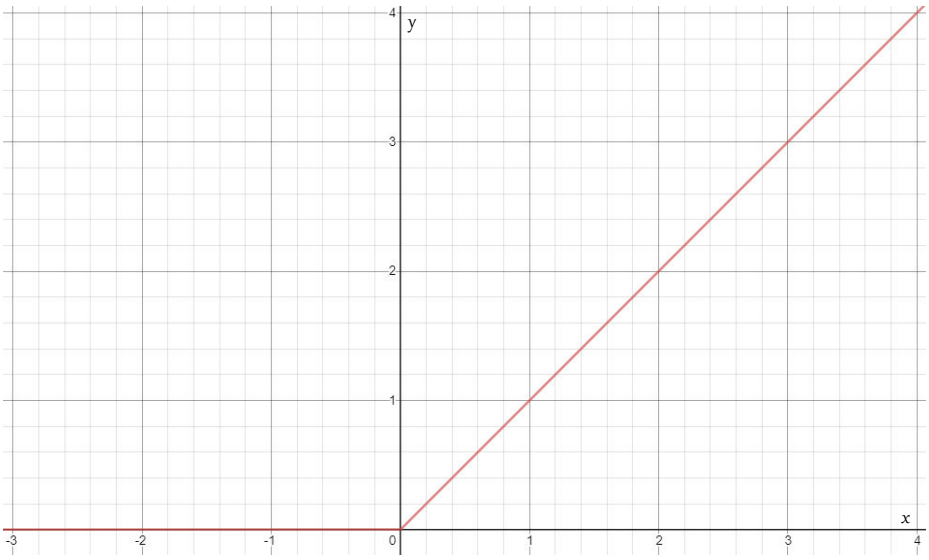
Obr. 1.6: Graf funkce Tanh.

4. **ReLU** *(Rectified Linear Unit)*
   Tato aktivační funkce je nelineární a je nejvíce používanou funkcí neuronových

sítí. Matematicky je popsána jako

$$F(x) = max(0, x) \tag{1.7}$$

z čehož vyplývá, že pokud je vstup menší než nula, je nastaven na nulu. Tuto vlastnost lze lépe vidět na grafu 1.7. Její hlavní výhodou je, že při použití nejsou aktivovány všechny neurony ve stejný čas. To znamená, že neuron bude deaktivován pouze tehdy, když výstup lineární transformace bude roven nule. ReLU je efektivnější než ostatní právě díky této selektivní aktivaci neuronů. Další výhodou jsou také rychlejší výpočty, protože nemusí provádět dělení a mocnění. Používá se velmi často ve skrytých vrstvách neuronových sítí. Její hlavní nevýhodou je ale její „křehkost“ při trénování, kdy způsobuje zánik některých gradientů a tím i neuronů. Tyto neurony jsou následně považovány za nulové. Problém s tzv. „mrtvými neurony“ řeší vylepšená funkce s názvem „*Leaky ReLU*“ [21, 22].



Obr. 1.7: Graf funkce ReLU.

5. **Leaky ReLU**

   Jak bylo zmíněno výše, jedná se o vylepšenou verzi základní aktivační funkce ReLU. U této funkce je problém neaktivních neuronů vyřešen tak, že v případě příchodu záporné hodnoty není nastavena nula jako v předchozím případě, ale je nastavena velmi nízká hodnota tak, aby nebyla nulová. Tímto se neurony udrží v aktivním stavu v celém procesu propagace (šíření informací v síti). Matematicky lze tuto funkci vyjádřit jako

$$F(x) = \alpha x + x, \tag{1.8}$$

kde koeficient α je nastaven na $0,01$. Graficky je tak viditelná narůstající záporná složka, která pro neurony rozšiřuje použitelné konstanty. Poprvé byla tato funkce představena v roce 2013 a použita na datové množině pro automatické rozpoznání řeči [21, 22].

Obr. 1.8: Graf funkce Leaky ReLU s konstantou $\alpha = 0,01$.

6. **Softmax**
   Funkce Softmax je kombinací několika funkcí typu sigmoid. Jak bylo zmíněno, funkce Sigmoid na svém výstupu vrací hodnoty v intervalu $\langle 0, 1\rangle$, kdy tyto body mohou být vnímány jako body představující jednotlivé pravděpodobnosti. Softmax vykonává stejnou činnost, čímž pro každý datový bod určuje pravděpodobnost určité třídy (je ji tak možné považovat za ekvivalent distribuční funkce). Na rozdíl od funkce Sigmoid, která je používána k řešení problémů binární klasifikace, je Softmax využívána pro klasifikaci více tříd. Matematicky může být hodnota funkce vypočítána podle

$$F(x) = \frac{\exp(x_i)}{\sum_j \exp(x_i)}. \tag{1.9}$$

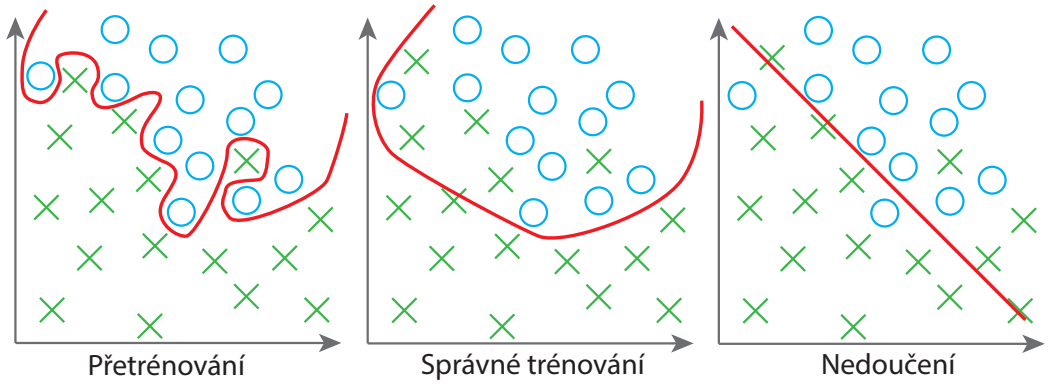
   Při použití funkce je vzhledem k její povaze nutné dodržet pravidlo počtu neuronů v poslední vrstvě neuronové sítě, kterých by pro správnou klasifikaci měl být stejný počet, jako je počet tříd datové množiny [22].

1.2.4 Přetrénování a nedoučení

Po uvedení způsobu trénování pomocí aktualizace neuronů, optimalizačních algoritmů a aktivačních funkcí je vhodné zmínit, jaké nejčastější problémy mohou při trénování nastat. Prvním z nich je tzv. přetrénování *(overfitting)*.

Přetrénování je jedním z nejčastějších problémů při trénování neuronových sítí. Vzniká v trénovací fázi procesu a vyznačuje se tím, že v určitém čase při trénování dosáhne určité (maximální) přesnosti a už není schopna se dále trénovat k řešení problémů. Neuronová síť se tak pouze učí náhodné souvislosti z trénovacích dat. Tento problém je ekvivalentem empirického pozorování změny chyby na testovacích datech. Schopnost generalizace neuronové sítě dosahuje nejlepších výsledků před tím, než tato chyba na testovacích datech začne narůstat. Přetrénování tak nastává, když model popisuje náhodnou chybu nebo datový šum místo zásadních souvislostí.

Druhým problémem, který je sice méně častý, ale dokáže také způsobit nepřesnost či neschopnost klasifikace, je nedoučení *(underfitting)*. Nedoučení je opakem přetrénování a projevuje se tím, že model není schopen zachytit proměnnost dat. Jinými slovy, model je příliš jednoduchý a nedostatečně robustní, aby mohl být použit na zvolená data. Oba dva tyto problémy jsou znázorněny na obrázku 1.9, který obsahuje jednotlivé datové body dvou tříd (binární klasifikace). Při přetrénování lze vidět, že výsledný model je až příliš přesný na to, aby „to mohla být pravda“. Ve skutečnosti ale není schopen klasifikovat na testovacích datech, tzn. na těch, které model „neviděl“. Při nedoučení je zase z obrázku vidět, že model je příliš jednoduchý a nedoučený na rozdělení většiny datových bodů do správných tříd [23]. Pro to,



Obr. 1.9: Případy trénování.

aby model nebyl přetrénován nebo nedoučen se využívá dvou typů metod.

1. **Postihové metody** *(penalty)*
   Používají se pro vyhnutí se přetrénování. Pro vysvětlení je možné uvést jednoduchý příklad: Chyba trénovací množiny je označena jako $E_{trénovací}$ a chyba testovací množiny jako E_{test}. Cílem je najít veličinu h, která minimalizuje E_{test}. Problémem je, že nelze přímo zhodnotit E_{test}, lze změřit $E_{trénovací}$, ale hodnoty jsou příliš optimistické. Proto

$$E_{test} = E_{trénovací} + \text{postih}, \tag{1.10}$$

kde je přímo penalizována komplexnost modelu. Lze ji také vyjádřit jako

$$E_{test} = E_{trénovací} + \lambda, \tag{1.11}$$

kde λ (komplexnost modelu) je stěžejní veličinou ovládající vlastnosti modelu. Pomocí tohoto parametru je tak možné uměle měnit odsazení [23].

2. **Metody dřívějšího zastavení trénování** *(early stopping)*
   Tyto metody se používají při přetrénování a nedoučení, kdy každé části datové množiny (trénovací, validační a testovací) je přiřazeno kritérium pro zastavení průběhu, čímž je docíleno určité přesnosti před tím, než se síť začne přetrénovávat. Závisí tak pouze na uživateli, který experimentálně určí moment zastavení průběhu (například pomocí funkce `EarlyStopping`, která se v jazyce Keras řadí mezi tzv. *callbacky*) [23].

2 Současný stav vědy a techniky

V posledních letech se oblast detekce a klasifikace objektů pomocí strojového učení exponenciálně rozrostla. Díky dostupným nástrojům, které jsou zdarma je možné si i za amatérských podmínek natrénovat vlastní model k jakékoliv činnosti. Za předpokladu dostatku vhodných tak mohou být aplikace velmi rozmanité. Pro použití na komplexnější úkoly jako například rozpoznávání řeči nebo ručně psaného písma je spíše vhodné použít rekurentní neuronové sítě [24]. Pro zpracování obrazu se používají konvoluční neuronové sítě, které jsou v moderní vědě hojně využívány.

2.1 Současná řešení

Článek [25] se zabývá vytvořením systému pro detekci a počítání vozidel na dálnicích s použitím hlubokého učení. Popisuje metody, kterými bylo dosaženo vytvoření klasifikačního modelu a použitou datovou množinu, která obsahovala přes 11 tisíc vzorků. V práci je použita technika segmentování, která si klade za cíl výrazně zpřesnit klasifikační výsledky. Vzhledem k tomu, že data z dopravních kamer jsou těžko dostupná a zřídkakdy zveřejňována, tak se práce pečlivě věnuje výběru datové množiny, která by poskytla neuronové síti robustní základ. Použity byly celkem čtyři datové množiny (PASCAL VOC, ImageNet, COCO a poslední, která byla vytvořena autory tohoto řešení).

Postupem práce pro klasifikaci je vložení dat z reálného provozu. Poté je za pomocí grafických filtrů a maskování z dat oddělena část povrchu dálnice od vozidel a data jsou předložena k natrénování neuronové síti YOLOv3. Dále je zde podrobněji popsána příprava dat ke zpracování neuronové sítě (tzv. *preprocessing*) a algoritmy použité ke sledování objektů v obraze. Výsledkem je model, který je natrénován na tři typy vozidel: automobily, autobusy a nákladní automobily. Rozdělení datové množiny označují autoři za „klasické“ – 80 % trénovací část a 20 % testovací část. Vzhledem k tomu, že výsledný model obsahoval falešně pozitivní detekce, tak pro potřeby této publikace byla výsledná přesnost klasifikace modelu vypočítána, aby byla minimalizována chyba. Nakonec se autoři věnují určování směru jedoucích vozidel, vytvoření virtuálního souřadnicového systému a jeho převedení do kartézské soustavy souřadnic pro výpočet přibližné rychlosti vozidel, která ale není pro tuto práci stěžejní tak, jako popsaná detekce a počítání vozidel. Automobily byly detekovány s přesností $86,46$ %, autobusy s přesností $88,57$ % a nákladní automobily s přesností $88,61$ %.

Druhým řešením, které tématicky úzce souvisí s cílem této práce, je publikace [26]. V ní se autoři taktéž zabývají detekcí vozidel, ale s jinými typy vozidel jako například terénní automobily nebo minibusy. Oproti předchozí práci je trénování

prováděno s menší datovou množinou, která čítá zhruba pět tisíc trénovacích vzorků, kde největší část je zastoupena klasickými automobily. Trénovací a testovací množiny jsou u každé kategorie rozděleny v poměru zhruba 50 % na 50 %. Použity jsou modely neuronových sítí VGG16, ZF a ImageNet. V úvodu experimentu je provedena prvotní inicializace modelem ImageNet sítě k natrénování nezávislé RPN sítě. Poté je inicializován model Fast-RCNN sítě váhami z předešlého trénování modelu RPN a návrh předešlé RPN sítě je použit jako vstup do Fast-RCNN sítě k jejímu natrénování. Znovu jsou inicializovány parametry RPN sítě za použití parametrů Fast-RCNN sítě a RPN síť je doladěna. Nakonec proces iterativně trénuje RPN a Fast-RCNN sítě. Datová množina je převzata z MIT a Caltechu, které mají vlastní datové množiny vozidel. Celkem byly testovány dvě sítě (ZF a VGG16), kdy nejlepších výsledků dosahovala ve všech třech kategoriích síť VGG16. Té se podařilo klasifikovat automobil s přesností 82, 3 %, minibus 74, 8 % a SUV 70, 1 %. Na závěr byla porovnána i rychlost klasifikace modelů sítě Fast-RCNN, Faster-RCNN a modelu, vytvořeného v tomto článku, kdy vytvořený model dosahoval 4× rychlejší klasifikace oproti Fast-RCNN a 1, 5× rychlejší klasifikace oproti sítí označované jako Faster-RCNN.

Třetí práce [27] se věnuje detekci objektů za pomoci vzdušných bezpilotních letounů (UAV) v reálném čase pomocí hlubokého učení běžícího na platformě Android, a tím s touto prací úzce souvisí. Jedná se o zpracování videa v reálném čase, které je do mobilního zařízení s Androidem přenášeno přes bezdrátový signál z dronu. V rámci této práce byla vyvinuta jednoduchá Android aplikace, která přijímá snímky z dronu a zobrazuje je uživateli. Klasifikační operace běží na pozadí při dekódování přijaté sekvence snímků. Proces klasifikace trvá zhruba 300 milisekund. Pro detekování objektů byly použity předtrénované modely sítí Inception-V3 a Mobilenet. Ve výsledku je pak možné klasifikovat objekty patřící až do tisíce kategorií.

2.2 Konvoluční neuronové sítě

Při zpracovávání obrazu je využíváno mnoho technik (algoritmů), jak docílit porozumění obrazu strojem. Jak bylo zmíněno, strojové učení zahrnuje oblast hlubokého učení pomocí neuronových sítí. Pojem hluboké učení odkazuje na větší hloubku neuronových sítí. Tato hloubka se projevuje hlavně ve skrytých vrstvách sítě, které mohou být pro zpracování obrazu tvořeny převážně konvolučními vrstvami. Konvoluční neuronové sítě je tak možné vnímat jako podmnožinu hlubokého učení.

Příkladem, na kterém je možné si vysvětlit obecnou funkci konvolučních neuronových sítí, bude v tomto případě obrázek 2.1. Klasifikační model má k dispozici vstupní data – jeden obraz, který musí být přidělen do jedné z daných kategorií, například `"automobil","autobus","jizdni_kolo"`. Vzhledem k tomu, že stroj

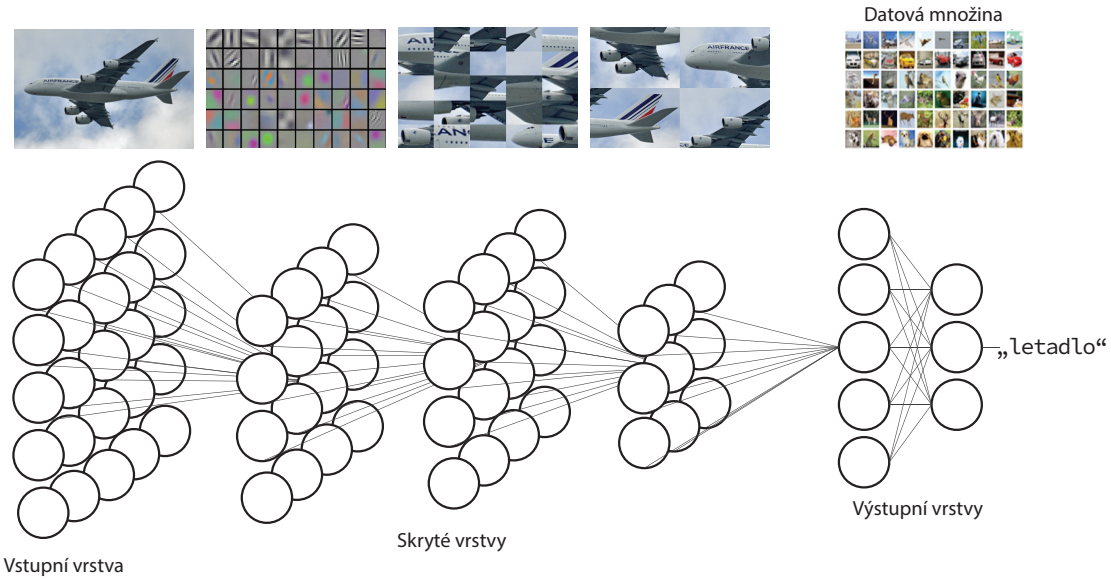
Obr. 2.1: Rozložení obrazu na tenzor hodnot.

nemůže nijak určit kategorii bez předchozího naučení, musí data projít několika vrstvami trénování. Na obrázku 2.1 je vidět rozložení malé plochy obrazu na prostorovou matici hodnot, se kterou konvoluční sítě pracují a přes které se dostávají k požadovaným a zapamatovatelným informacím. Obraz má velikost 650×490 pixelů, kromě těchto rozměrů má navíc i svou barevnou hloubku, to znamená další tři hodnoty navíc (podle barevného modelu RGB – červená, zelená a modrá). Pro zpracování tohoto konkrétního obrazu je tak zapotřebí neuronovou sítí analyzovat $650 \times 490 \times 3 = 955500$ hodnot. Nejde tak pouze o matici hodnot, ale o trojrozměrný tenzor, který je konvoluční neuronovou sítí zpracováván. Cílem konvoluční neuronové sítě je klasifikovat těchto téměř milion hodnot tak, aby uživateli bylo na výstupu k dispozici přiřazení obrazu do jedné kategorie, v tomto případě `"autobus"` [28].

2.2.1 Učení podle příznaků

Pro hlubší vysvětlení, jak konvoluční neuronové sítě fungují je třeba zajít do jejich skrytých vrstev. Pro obecnou klasifikaci s cílem predikce, která bude využita v praktické části této práce, se využívá kombinací několika úrovní struktury neuronové sítě. První úrovní je vstupní vrstva, která má většinou za úkol data sjednotit a předzpracovat. Následuje několik skrytých vrstev a síť končí výstupní vrstvou nebo více výstupními vrstvami. V každé úrovni neuronové sítě se při zpracování obrazu tvoří příznaky, které jsou jednotlivými vrstvami extrahovány. Tyto příznaky postupují dále, hlouběji do sítě, kde je z příznaků podle teorie neuronových sítí sestaven další, už pokročilejší příznak. Výsledkem je přiřazení obrazu do správné kategorie

právě podle těchto signálových příznaků. Sestavený celek se pak nazývá „příznaková mapa“. Tento proces je znázorněn na obrázku 2.2 [29]. Příznaková mapa je v její



Obr. 2.2: Klasifikace pomocí příznakové mapy.

nejranější fázi tvořena pouze malými příznaky, ze kterých se po jejich spojení stává pro člověka čitelnější obraz. Tento základ je velmi důležitý, protože se od něj později odvíjí seskupování větších příznaků. Příznaky jsou získávány pomocí signálových filtrů, které zaznamenávají vlnovou délku λ, orientaci θ, fázový posun ϕ, poměr velikosti γ a šířku pásma b. Impulzní odezva těchto filtrů, které se nazývají gabor filtry, je definována jako sinusoida vynásobena Gaussovskou funkcí. Filtr má reálnou a imaginární složku reprezentující ortogonální směry [30]. Pro využití v konvolučních neuronových sítích se pro extrakci příznaků z obrazu používá sady gabor filtrů s odlišnými frekvencemi a směry, které jsou dvojrozměrné (2D) a diskrétně mohou být vyjádřeny jako

$$G_c[\mathrm{i},\mathrm{j}] = B\mathrm{e}^{-\frac{(\mathrm{i}^2+\mathrm{j}^2)}{2\sigma^2}}\cos(2\pi f(\mathrm{i}\cos\theta + \mathrm{j}\sin\theta)), \tag{2.1}$$

$$G_s[\mathrm{i},\mathrm{j}] = C\mathrm{e}^{-\frac{(\mathrm{i}^2+\mathrm{j}^2)}{2\sigma^2}}\sin(2\pi f(\mathrm{i}\cos\theta + \mathrm{j}\sin\theta)), \tag{2.2}$$

kde B a C zastupují normalizující faktory ke zjištění, f je hledaná frekvence v textuře obrazu, θ umožňuje najít konkrétně orientovanou texturu (nastavitelná) a parametr σ definuje rozšířitelnou odezvu filtru. Lze jím nastavit velikost analyzované oblasti v textuře. Parametry B a C jsou například v práci [31] nastaveny podle podmínek

$$\sum_{\mathrm{i}}\sum_{\mathrm{j}} G_c[\mathrm{i},\mathrm{j}]\cos(2\pi f(\mathrm{i}\cos\theta + \mathrm{j}\sin\theta)) = 1, \tag{2.3}$$

$$\sum_{\mathrm{i}}\sum_{\mathrm{j}} G_s[\mathrm{i},\mathrm{j}]\sin(2\pi f(\mathrm{i}\cos\theta + \mathrm{j}\sin\theta)) = 1, \tag{2.4}$$

aby se oběma dvěma směry (i a j) šířily stejně.

2.2.2 Funkce konvoluční neuronové sítě

Neurony v konvoluční neuronové síti se od neuronů v klasické neuronové síti liší tím, že jsou uspořádány v 3-rozměrném prostoru (šířka, výška a hloubka), jak je vidět na obrázku 2.2. Perceptrony v konvolučních vrstvách jsou propojeny pouze s menší oblastí předchozí vrstvy místo toho, aby byly plně propojeny (každý neuron s každým). Na konci architektury konvoluční neuronové sítě se redukuje plný obraz do jednoho vektoru, které zastupuje skóre jednotlivých tříd. Bližší vysvětlení funkce je možné si uvést na následujícím příkladu konvoluční neuronové sítě a obrazu z datové množiny s názvem CIFAR-10. Architektura sítě pro klasifikaci je jednoduchá a řídí se posloupností: vstupní vrstva, konvoluční vrstva, aktivační vrstva (ReLU), podvzorkovací vrstva a nakonec plně propojená vrstva ve výstupní části. Typy konvolučních vrstev jsou podrobněji vysvětleny níže.

- **Vstupní vrstva** *(input layer)*
  Na vstupu je v případě datové množiny CIFAR-10 obraz o rozměrech $32 \times 32 \times 3$. Tato vrstva drží informace o surovém obrazu, konkrétně o hodnotách jeho pixelů. V tomto případě se jedná o obraz s šířkou 32 pixelů, výškou 32 pixelů a hloubkou tří barevných kanálů (RGB).
- **Konvoluční vrstva**
  Konvoluční vrstva má za úkol spočítat výstup neuronů, které jsou napojeny na lokální oblasti vstupu. Každá vrstva počítá skalární součin mezi jejich váhami a malou oblastí, ke které jsou připojeny na vstupu. V konvolučních vrstvách se provádí aplikace konvolučního filtru na obraz, jehož výstupem je hodnota do příznakové mapy.
- **Aktivační vrstva – ReLU**
  ReLU vrstva aplikuje aktivační funkci $max(0, x)$ začínající na nule. Tím je velikost množství dat $32 \times 32 \times 3$ ponecháno beze změny. Funkce ReLU se používá pro zanesení nelinearit do modelu.
- **Podvzorkovací vrstva** *(pooling layer)*
  Tato vrstva má za úkol podvzorkovat data, to znamená zmenšit jejich rozměry tak, aby bylo zachováno co možná nejvíce důležitých informací pro konvoluční neuronovou síť. Sníží se tím výpočetní čas a využití paměti. Data jsou tak zmenšena na $16 \times 16 \times 12$, což je výsledek ponechání nejvyšší hodnoty textury a zahození ostatních hodnot.

- **Plně propojená vrstva**
  Poslední, výstupní plně propojená vrstva počítá hodnocení tříd (skóre) s výsledkem datového objemu $1 \times 1 \times 10$, kde každé z deseti čísel koresponduje se skóre dané třídy (tzn. 10 tříd v datové množině CIFAR-10). Každý neuron v této vrstvě je propojen s každou hodnotou předchozího objemu dat (s každým neuronem předešlé vrstvy).

Z tohoto příkladu je vidět, že konvoluční neuronové sítě přetvářejí originální obrazovou vrstvu s originálními surovými hodnotami do třídy obsahující skóre. Některé vrstvy ale neobsahují trénovatelné parametry. Konkrétně konvoluční a plně propojená vrstva provádí transformace, které jsou ne jen výsledkem aktivace neuronů, ale také parametrů (váhy a odsazení neuronů). Na druhou stranu ReLU a podvzorkovací vrstvy implementují danou (fixní) funkci. Parametry těchto vrstev jsou trénovány pomocí gradientního sestupování tak, aby skóre tříd bylo konzistentní s označením v trénovací množině každého obrazu [32].

2.2.3 Přenesené učení

Při učení konvolučních neuronových sítí často dochází k problému nedostatku dat v datové množině, nebo i celé datové množiny. Z tohoto důvodu se využívá skutečnosti, že existují již natrénované robustní modely, na kterých se podílí celá komunita odborníků i běžných uživatelů. Tyto modely se natrénují na velké datové množině (většinou se jedná o data v řádech milionů vzorků a stovek až tisíců tříd) a poté se tento model použije jako inicializační pro oblast, která daného uživatele zajímá. Výraz „přenesené učení“ *(transfer learning)* tak zastává svého znění právě kvůli tomuto postupu, kdy se model přenese a dotrénuje podle konkrétních požadavků. Je možné se setkat se třemi hlavními scénáři, u kterých je vhodné využít přeneseného učení:

1. **Fixní model k extrakci příznaků**
   Tato možnost umožňuje využít již předtrénovaný model pouze k extrakci příznaků z nové datové množiny. Z modelu je možné odstranit poslední plně propojenou vrstvu, která obsahuje klasifikační skóre počtu tříd a zbytek konvoluční neuronové síť použít jako „extrahovač“ příznaků nové datové množiny. Jsou vypočítány vektory pro každý obraz, obsahující aktivace skrytých vrstev hned před klasifikátorem. Tyto příznaky jsou nazývány *CNN codes*. Jakmile jsou tyto kódy (příznaky) extrahovány pro všechny obrazy, trénuje se lineární klasifikátor na nových datech (tzn. za použití funkce Linear SVM nebo Softmax).
2. **Dolaďování modelu**
   Druhým možným přístupem není pouze nahrazení a přetrénování klasifikátoru

nad modelem nové datové množiny, ale taky doladění vah předtrénované sítě pokračováním procesu zpětné propagace (zpětného učení). Je možné doladit všechny vrstvy sítě nebo nechat některé beze změny (kvůli přetrénování) a doladit pouze některé části sítě. Používá se z toho důvodu, že počáteční vrstvy obsahují spíše zdrojovější funkce na extrakci příznaků (například detektory hran nebo zanedbatelné detektory barev). Tyto funkce jsou sice užitečné pro mnoho úkolů, ale v pozdějších vrstvách se stávají více citlivé na detaily tříd v originální datové množině. Například pro datovou množinu, která obsahuje jemnější kategorie (pes – rasa) se tak síť specializuje více na rasu. Vše ale závisí na tom, jaký je požadavek uživatele.

3. **Předtrénované modely**
   K tomuto řešení se přistupuje z důvodu dlouhého času učení na velkých datových množinách. Trénování na nich zabírá dny až týdny, a to i přes více vláknové zpracovávání (použití více kusů hardwaru). Z tohoto důvodu se zveřejňují tzv. *checkpointy*, které zastupují aktuální stav trénování a jsou k dispozici pro další nezávislé trénování a dolaďování [33].

2.2.4 Předtrénované modely – YOLO

V dnešní době je možné z různých zdrojů stáhnout předtrénované modely konvolučních neuronových sítí, které tak část práce odvedou za uživatele. Vzhledem k tomu, že tvoření konvoluční neuronové sítě tak, aby dosahovala přesností přijatelných pro nasazení v reálném provozu, je většinou experimentální a tím i velmi časově náročné, je možné těchto předtrénovaných modelů využít. Ke stažení je jich mnoho, vždy je ale nutné dodržovat pokyny v dokumentaci a teoreticky se na konkrétní model připravit. Většinou jsou předtrénované modely rozsáhlejším projektem vědeckých pracovníků, kteří po dosažení nejlepší možné přesnosti modelu publikují svoji práci ostatním. V této práci jsou nejprve využity předtrénované modely sítě YOLO *(You Only Look Once)*, která je s postupem času upravována a povyšována na další verze. První verzí, se kterou bude prováděn experiment, je verze YOLOv3. Volně dostupné jsou také další dvě verze – YOLOv4 a YOLOv5. Níže jsou popsány jejich vlastnosti.

1. **YOLOv3**
   YOLOv3 byla uvedena v roce 2018 Josephem Redmonem a Alim Farhadim publikací [34], která je sice neformálně psaná, ale přes to informačně plná. Při klasifikaci je kolem objektů vygenerována obdélníková hranice, kdy síť předpovídá 4 souřadnice pro každé ohraničení. Toto ohraničení je výsledkem skóre, které je počítáno pomocí logistické regrese. Dále je podrobněji popisováno funkce generování ohraničení kolem objektů. Klasifikace je možná na datech

ve více třídách, kdy ale není použita aktivační funkce Softmax z důvodu jejího vlivu na přijatelný výkon modelu. Model byl použit na datovou množinu *Open Images Dataset*, která obsahuje mnoho překrývajících se značek objektů (například značka „žena“ a „osoba“ v jednom ohraničení). Funkce Softmax ale předpokládá, že jedno ohraničení patří jednomu objektu, vznikal by tak problém s klasifikací. Místo ní jsou použity logistické klasifikátory. Konvoluční architektura modelu obsahuje 53 konvolučních vrstev a byla nazvána *Darknet-53*. YOLOv3 byla otestována na datové množině COCO, která bude použita v prvním experimentu a její mAP metrika, která zastupuje střední průměrnou hodnotu všech tříd dohromady pod trénovací křivkou, je 57, 9 % při 20 snímcích za vteřinu. Pro srovnání, například síť RetinaNet dosahovala mAP 57, 5 %, ale při 5 snímcích za vteřinu [34]. Při porovnávání rychlostí sítí je užíváno jednotky FPS *(Frames per Second)*, kdy vše nad 30 FPS je považováno za klasifikaci v reálném čase. Při klasifikaci nad 30 FPS (35 FPS) dosahovala síť YOLOv3 přesnosti 55, 3 %.

2. **YOLOv4**
   YOLOv3 se dočkala vylepšení až v dubnu roku 2020, kdy byla publikována síť YOLOv4. Její architektura po vstupní vrstvě obsahuje model CSPDarknet-53, následuje model SPP a PAN a končí modelem YOLOv3. Je tak kombinací několika předem publikovaných modelů. Její mAP dosáhla na COCO datové množině 65, 7 % při 23 FPS a přesnosti 64, 9 % při 31 FPS [35].
3. **YOLOv5**
   K modelu YOLOv5 zatím nebyla vydána žádná oficiální publikace. YOLOv5 je model založen na knihovně PyTorch a je stále ještě ve vývoji. Jako obvykle se ale od něj zvýšení přesnosti klasifikace zároveň s dalšími metrikami, jako je mAP a snímková frekvence [36].

2.2.5 Datové množiny

Výběr datové množiny pro trénování je zásadní pro následný výkon modelu, a proto by neměl být podceňován. Jedná se o natrénování modelu na vzorcích, které v datech musí být zastoupeny v dostatečném množství, což bývá jedním z nejčastějších problémů. Proto se využívá různých technik, jako je například umělé rozšiřování dat *(data augmentation)*, kdy je datová množina uměle obohacena sice o obsahově stejné prvky, ale jakýmkoliv způsobem transformované. Tyto vzorky tak tvoří úplně nový obraz za použití jednoduchých úprav (například otočení nebo odstranění pozadí) [37].

Dostupných datových množin je v dnešní době už mnoho, kdy každá datová množina má svá specifika. Množiny jsou vybírány k účelu, ke kterému mají být po-

užity, například rozpoznávání obličejů, činností, ručně psaného písma, zvuků nebo biologických dat. V první části experimentu bude použita datová množina s názvem „*Microsoft COCO: Common Objects in Context*“ publikována v roce 2015 a neustále aktualizována až doposud. Také bude využit předtrénovaný model s váhami datové množiny ImageNet.

- **Microsoft COCO: Common Objects in Context**
  Datová množina COCO je zdrojem pro mnoho modelů konvolučních neuronových sítí, se kterými je možné se v praxi setkat. Je možné na ní natrénovat algoritmy pro segmentaci objektů nebo rozpoznávání objektů z kontextu. Obsahuje přes 330 tisíc obrazů, kde více než 200 tisíc je označených třídou, do které patří. Je možné si vybrat až z 80 kategorií objektů. Podle oficiálních porovnávacích testů bylo na této množině dosaženo největší přesnosti 51 % [38], a to v publikaci [39], která se zabývá detekcí objektů.
- **ImageNet**
  Datová množina ImageNet je organizována s podobnou hierarchií, jako množina WordNet, která obsahuje pouze podstatná jména. Tato hierarchie popisuje každý uzel, ve kterém se nachází tisíce obrazů. Momentálně obsahuje jeden uzel v průměru 500 obrazů. Každý koncept v množině WordNet je popsán souslovím, které se nazývá „synonym set“ nebo zkráceně „synset“ a v datové množině jich je více než sto tisíc. V množině ImageNet je zhruba tisíc obrazů popisující každý synset. U obrazů je dbáno na zachování jejich kvality a každý je ručně označen člověkem. Největší přesnosti se podařilo s touto datovou množinou dosáhnout týmu ze společnosti Google, a to během roku 2020. Přesnost činí $88,55$ % [40].
- **MNIST**
  Databáze MNIST *(National Institute of Standards and Technology)* je rozsáhlá datová množina, obsahující ručně psané číslovky. Její trénovací množina obsahuje 60 tisíc vzorků a testovací 10 tisíc vzorků. MNIST je podmnožinou databáze NIST, která je ještě rozsáhlejší, ale pro rozpoznávání písma příliš zašuměná. Vzhledem k její jednoduchosti je tato množina vhodná i pro začátečníky se strojovým učením, protože není potřeba vkládat čas do předzpracování a formátování obrazů – velikost každého vzorku je normalizována a každý vzorek je vycentrován. Nejmenší chybovosti bylo dosaženo v práci zabývající se regularizací neuronových sítí pomocí zahazování vah – $0,21$ % [41, 42].
- **CIFAR-10**
  Datová množina CIFAR-10 je podmnožina databáze s názvem „*80 million tiny images*“, která byla 29. 6. 2020 stáhnuta z důvodu údajného shledání některých obrazů urážlivých. Vzhledem k velkému počtu vzorků by manuální prohledávání s cílem najít „závadná“ data bylo téměř nemožné, proto byla

pro veřejnost smazána. CIFAR-10 obsahuje 60 tisíc barevných obrazů o rozměrech 32×32 pixelů v deseti třídách. Každá třída tak obsahuje 6 tisíc vzorků. Databáze je rozdělena na 50 tisíc trénovacích a 10 tisíc testovacích obrazů. Nejvyšší přesnosti bylo dosaženo v práci [43] v květnu roku 2020 týmem vědců ze společnosti Google, a to $99,37$ % [44].

2.3 Detekce na mobilních zařízeních

Jak bylo zmíněno na začátku teoretického úvodu této práce, klasifikace a detekce na mobilních zařízeních je využívána dnes již téměř všemi výrobci chytrých mobilních telefonů. Pro co nejpřesnější a nejrychlejší výsledky detekce objektů v reálném čase jsou ale pro tyto úlohy vyhrazovány samostatné části výpočetních jednotek v mobilním zařízení. Je také využíváno mnoha různých technik a knihoven k detekci obrazů, tato práce se ale zabývá použitím konvolučních neuronových sítí v mobilním zařízení.

2.3.1 Klasifikace vs. detekce

V analýze obrazu pomocí konvolučních neuronových sítí je možné se setkat se třemi pojmy jeho zpracování. Těmito pojmy jsou klasifikace, detekce a segmentace.
Při předložení obrazu natrénované neuronové síti, kde se nachází pes je určeno, že na tomto obrazu (v této instanci) se nachází pes. Tímto klasifikace končí. V případě, že je na obrazu například další zvíře než jen pes, může dojít ke zmatečným výsledkům. V tomto případě je vhodné natrénovat klasifikátor, který obsahuje více než jednu značku. I přes to ale stále není možné zjistit, kde na obrazu se objekty nachází. Pojem „detekce objektu“ by ale tuto schopnost měl mít a na rozdíl od klasifikace ukázat přesné místo v obrazu, kde se rozpoznaný objekt nachází. Pro uživatele je vhodné tento objekt označit například obdélníkem (tzv. *bounding box*). Lokalizaci objektů je tak možné rozdělit do dvou částí:

- **Klasifikace objektu**
  Klasifikace objektu zahrnuje techniky, které mohou být kategorizovány jako parametrické nebo neparametrické a techniky učení s učitelem nebo bez učitele. Při klasifikaci s učitelem jsou poskytovány výsledky na základě rozhodovací hranice, která většinou závisí na poskytnutém vstupu a výstupu modelu při jeho trénování. Při technice klasifikace bez učitele jsou poskytovány výsledky na základě analýzy vstupních dat (datové množiny), tzn. příznaky nejsou do modelu přímo dodávány. Hlavními kroky při klasifikaci jsou výběr vhodného klasifikačního systému, výběr vhodných trénovacích vzorků, předzpracování dat a výběr vhodné klasifikační metody, extrakce příznaků, zpracování dat po

klasifikaci a zhodnocení přesnosti. V této technice je vstupem obraz a výstupem je jeho predikovaná třída. Příkladem může být binární klasifikace, jejímž cílem je správně přiřadit data do dvou tříd nebo více-třídní klasifikace, která obsahuje více než dvě třídy. Klasifikace obrazů je zvolena pro klasifikaci vozidel v této práci. Nevýhodou klasifikace obrazů je časová náročnost během trénovací fáze. Není tak vhodná pro zpracování velkých objemů dat.

- **Detekce objektu**
  Problém detekce objektu spočívá v rozhodnutí softwaru, do které třídy objekt na obrazu patří (klasifikace) a kde přesně se nachází. Jednotlivé objekty se zpravidla rozdělují do samostatných instancí. V praxi to znamená, že na obrazu může být více objektů z více tříd. Technika detekce objektů je používána stále častěji v rychle rostoucích oblastech jako autonomní řízení, detekce obličejů nebo rozpoznávání státních poznávacích značek. Rychlý rozvoj je možný právě díky technikám hlubokého učení. Nevýhodou při detekci mohou být vzniklé nepřesnosti při detekování objektů v příliš zašuměném a tvarově komplikovaném prostředí [1, 45].

3 Experimentální část

Prakticka část této práce zahrnuje dvě hlavní části. První je navržení a vytvoření Android aplikace a druhou je trénování a testování modelů. Je vyzkoušeno několik přístupů trénování. Prvním je použití modelu SSD MobileNet a porovnání rychlosti detekce s již otestovanými mobilními zařízeními. Následuje natrénování vlastního modelu s vytvořenou datovou množinou, natrénování modelu s volně dostupnou datovou množinou a závěrem je dotrénování existujícího modelu o vlastní data. Model je následně vždy převeden do formátu kompatibilního s mobilními telefony s operačním systémem Android, kde bude probíhat detekce vozidel silniční dopravy ve vytvořené aplikaci.

Trénování modelů bude probíhat na grafickém čipu značky nVidia s typovým označením GeForce RTX 2070, disponující 2304 CUDA jádry s frekvencí 1620 MHz na jádro a pamětí 8 GB. Trénování lze také provést na centrálním procesoru, vzhledem k povaze jeho funkce by ale trénování bylo neefektivní [46, 47].

3.1 Tensorflow

Tensorflow je otevřená platforma založena na principu úplnosti *(end-to-end)* strojového učení. Obsahuje nástroje a knihovny pro komunitu, která se zabývá strojovým učením na nejmodernější úrovni. Umožňuje tak vývojářům vydávat celé aplikace strojového učení a v kombinaci s libovolným vysokoúrovňovým API *(Application Programming Interface)*, kterým může být například Keras, umožňuje implementaci strojového učení v mnoha různých odvětvích. Tensorflow je využíván společnostmi jako je Google, Airbnb nebo Twitter a je neustále aktualizován. První verze byla vydána firmou Google, konkrétně „Brain“ týmem, v roce 2017 a od té doby se Tensorflow stále rozrůstá. S příchodem a větším využitím dalších technologií pronikl Tensorflow v roce 2018 do jazyku JavaScript jako `Tensorflow.js`, v roce 2019 pak byla vydána jeho druhá verze – Tensorflow 2.0 a také verze TensorFlow Graphics pro hluboké učení v počítačové grafice [48].

3.1.1 Tensorflow Lite

Tensorflow Lite vznikl z platformy Tensorflow jako její derivát, který s ní úzce souvisí. Jeho úkolem je převádění natrénovaných modelů vytvořených na platformě Tensorflow tak, aby byla provedena jejich optimalizace z hlediska rychlosti a nároků na paměť. Tento převedný formát je poté možné využít pro detekci a klasifikaci v mobilních zařízeních s operačním systémem Android, iOS, Linux nebo Raspberry Pi.

Optimalizace je prováděna na uživatelem uloženém modelu, který obsahuje strukturu sítě, váhy, odsazení a aktivační funkce. Tato data jsou ve 32bitovém formátu, který může být zmenšen na 16 nebo 8bitový formát. Také je využito kvantizování dat, kdy jsou ponechány ty nejdůležitější informace modelu [49].

3.2 Keras

Keras je aplikační programovací rozhraní pro hluboké učení napsané v programovacím jazyce Python. Běží výhradně nad platformou Tensorflow, kterou pro svou správnou funkci využívá. Keras byl vyvinut se zaměřením na rychlost experimentů, to znamená schopnost vývojáře zrealizovat svůj nápad v co možná nejkratším čase. Podporuje spouštění výpočtů jak na CPU, GPU, tak i na TPU [50].

3.3 Android Studio

Android Studio je oficiální vývojové prostředí (IDE) pro vývoj aplikací platformy Android s podporou vývojového prostředí IntelliJ IDEA. Toto vývojové prostředí slouží k programování v jazycích Java, Groovy a dalších. Android Studio poskytuje vývojářům mnoho užitečných funkcí jako například kompilovací systém na základě nástroje Gradle, integraci s verzovacími nástroji nebo dynamické upravování kódu. Důležitá je také podpora virtuálních Android zařízení. Aplikaci je tak možné testovat i bez hardwarově připojeného mobilního telefonu [51].

3.4 Programovací prostředí Flutter

Programovací prostředí Flutter je otevřené *(open-source)* uživatelské rozhraní SDK pro vývoj softwaru na platformy Android, iOS, Windows, MacOS, Linux, Google Fuchsia a web. První verze Flutteru byla publikována v roce 2015 na vývojové konferenci Dart. Hlavními komponentami Flutteru jsou: Platforma Dart (programovací jazyk), Flutter modul (podporován Android Studiem), základní knihovna a designově specifické komponenty *(widgety)*. Velkou výhodou Flutteru je možnost časově nenáročné migrace kódu mezi jednotlivými platformami (například Android a iOS) [52].

3.5 Jupyter Notebook

Jupyter Notebook je webová aplikace, která zastupuje vývojové prostředí. Přístupem přes web nahrazuje konzolový přístup k interaktivnímu provádění výpočtů,

čímž zlepšuje přehlednost kódu, tvoření dokumentace, spouštění a diskuzi výsledků kódu. Kód je spouštěn z webové aplikace, což umožňuje bohatou reprezentaci výsledků například v HTML, LaTeX, PNG nebo SVG formátech. Jeho hlavní výhodou je možnost rozdělování kódu do buněk, které mohou být spouštěny nezávislé na sobě [53].

3.6 Vrstvy konvolučních neuronových sítí v knihovně Keras

Jak bylo zmíněno, v Kerasu je možné rychle a poměrně jednoduše tvořit vlastní neuronové sítě. Základními stavebními bloky pro konvoluční neuronové sítě jsou jednotlivé vrstvy, kde každá obsahuje vstupní a výstupní tenzorový výpočet a uložený stav v proměnné (váhy). Tyto vrstvy se tak chovají jako funkce. Níže jsou uvedeny názvy využívaných vrstev se stručným popisem jejich činnosti.

1. **Předzpracovací vrstva**
   Tato vrstva má za úkol připravit data na vstup do sítě. Pomocí ní je možné změnit velikost dat, obrazově je přiblížit, vycentrovat a ořezat nebo provést různě náhodné transformace. Praktickými příklady mohou být vrstvy `Resizing.layer`, `Rescaling.layer` nebo `CenterCrop.layer`.
2. **Konvoluční vrstva**
   Konvoluční vrstvy zajišťují konvoluci nad daty, to znamená operace jako například přechod konvolučního jádra nad obrazy. Jsou počítány hodnoty tenzorů a posílány dále do dalších vrstev. V Kerasu je možné najít vrstvy jako `Conv2D.layer`, `SeparableConv2D.layer` nebo `Conv2DTranspose.layer`.
3. **Podvzorkovací vrstva**
   Podvzorkovací vrstvy mají za úkol data podvzorkovat, jak bylo vysvětleno v teoretické části práce. Výrazně se tak snižuje počet trénovatelných parametrů sítě a tím procesní čas. V Kerasu jsou zastoupeny například vrstvami `MaxPooling2D.layer` nebo `AveragePooling2D.layer`.
4. **Vrstva jádra**
   Vrstvy jádra obvykle doplňují předchozí vrstvy. Jejich zástupcem je například vrstva `Dense.layer`, která obsahuje neurony umístěné hustě u sebe a provádí operaci: `výstup = aktivace(skalárníSoučin(vstup, jádro) + odsazení)`.
5. **Regularizační vrstva**
   Tato vrstva se používá jako doprovodná ke konvolučním, kdy její podvrstvy si kladou za cíl vyhnout se přetrénování modelu umělým nastavováním náhodně vybraných neuronů jako nečinné (nulové) při každé iteraci. Jedná se například o `Dropout.layer` nebo `SpatialDropout2D.layer` [54].

4 Výsledky

Tato kapitola se zabývá návrhem Android aplikace, která pomocí kamery detekuje objekty v reálném čase. Při spuštění aplikace probíhá přes videokameru a načtený model neuronové sítě klasifikace účastníků dopravy. Je vyzkoušeno několik možných přístupů. Prvním experimentem s prezentovaným výsledkem je porovnání rychlosti detekce předtrénovaného modelu na mobilním zařízení Samsung Galaxy A6+. Následuje série experimentů s různými množinami a neuronovými sítěmi.

4.1 Android aplikace

Pro návrh mobilní aplikace na platformě Android je použito vývojové prostředí Android Studio s modulem Flutter. Aplikace je testována na integrovaném emulátoru Android zařízení přímo ve vývojovém prostředí. Pro konečné testování v reálném provozu je použit mobilní telefon Samsung Galaxy A6+ z roku 2018. Mobilní telefon je k PC připojen přes rozhraní USB a po spuštění aplikace v Android Studiu je přístup v mobilním telefonu k aplikaci zajištěn přes nástroje pro tzv. *debugování*.

Aplikace je schopna otevřít kameru zadního fotoaparátu a v reálném čase detekovat objekty. Model je získán natrénováním neuronové sítě s využitím Tensorflow, Keras a Jupyter Notebook. Tento model je v surovém formátu (neoptimalizovaný), proto je uložen a pomocí skriptu níže převeden na verzi optimalizovanou pro mobilní telefony `.tflite`.

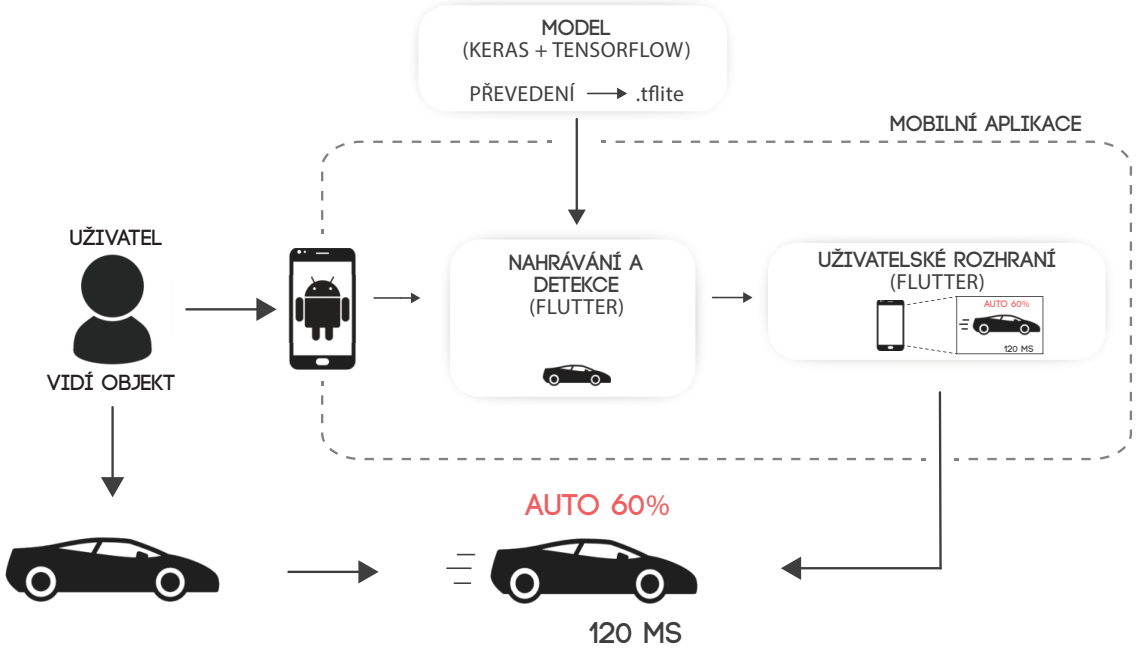
```
import tensorflow as tf
#Převedení modelu
converter = tf.lite.TFLiteConverter.from_saved_model
(cesta_k_souboru)
tflite_model = converter.convert()

#Uložení modelu
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

Výpis 4.1: Klíčový skript pro převod modelu.

Schéma návrhu aplikace je zobrazeno na obrázku 4.1. Struktura projektu je vygenerována automaticky vývojovým prostředím a je zobrazena ve složkové struktuře níže. Je tak možné začít tvořit vlastní aplikaci a není nutné se zabývat kompatibilitou a podpůrnými skripty. Klíčové složky pro aplikaci jsou vyznačeny červeně, pouze obsah těchto složek je upravován.

```
/
├── android
├── build
├── ios
├── test
├── lib
│   ├── HomePage.dart
│   ├── main.dart
│   └── MySplashPage.dart
└── assets
    ├── ...
    ├── model.tflite
    └── labels.txt
```

Aplikace se skládá ze tří tříd umístěných ve složce `lib` – spustitelná `main.dart`, úvodní strana `MySplashPage.dart` a hlavní strana `HomePage.dart`. Druhou hlavní složkou projektu je složka `assets`, která mimo jiné přílohy aplikace obsahuje soubor s převedeným modelem `model.tflite` a k němu patřičné značky `labels.txt`. Z této složky je model se značkami načítán příslušnou třídou k následné detekci objektů přes kameru.

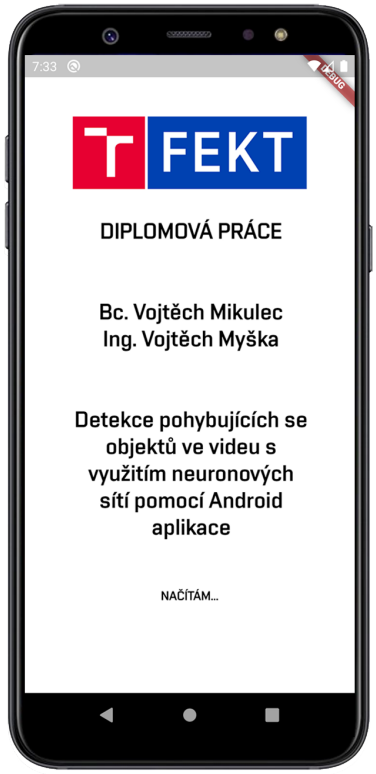


Obr. 4.1: Návrh aplikace pro detekci objektů.

4.1.1 Funkce tříd aplikace

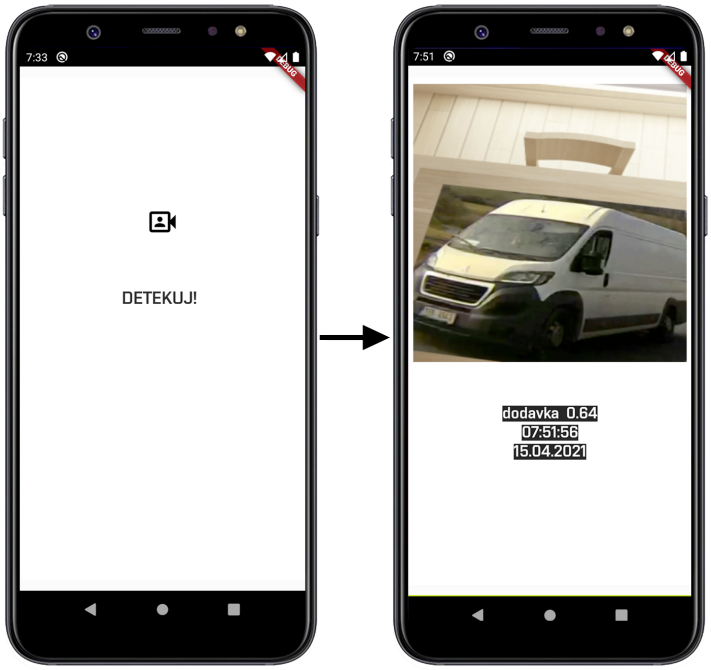
U vytvořené detekční aplikace je kladen důraz na jednoduchost. Neobsahuje žádné sledovací algoritmy, které by zatěžovaly hardware mobilního telefonu. Detekce objektu je vypsána u toho objektu, který má aktuálně nejvyšší pravděpodobnost detekce. Funkce a popis jednotlivých tříd je uveden níže.

- `main.dart`
  Třída `main.dart` je hlavní spustitelnou třídou aplikace. Vzhledem k tomu, že prostředí Flutter je založeno na větvení tzv. widgetů, tak obsahuje i kořenový widget pro celou aplikaci. Od tohoto widgetu se odvíjí všechny další založené widgety. Tato třída kontroluje dostupnost kamery fotoaparátu a odkazuje uživatele na třídu `MySplashPage.dart`. Je zde také možné upravit parametr pro zobrazení výkonnostních statistik aplikace.
- `MySplashPage.dart`
  Tato třída obsahuje widget, který uživateli zobrazí tzv. `SplashScreen` obrazovku. `SplashScreen` zobrazuje informace o aplikaci, logo, autora aplikace a vedoucího práce, jak je vidět na obrázku 4.2. Po dobu zobrazení této obrazovky je načítána další stránka (třída) aplikace, na kterou je uživatel přesměrován automaticky po stanovené době.



Obr. 4.2: Zobrazení úvodních informací o aplikaci.

- `HomePage.dart`
  Funkční třídou aplikace je třída `HomePage`. Jedná se o poslední část aplikace, do které je uživatel přesměrován. Při klepnutí na tlačítko kamery je spuštěna zadní kamera s detekcí objektů. V této třídě je načítán optimalizovaný `model.tflite` se značkami, je inicializována kamera a spuštěn proud obrazů, na kterém je následně prováděno vyhodnocování modelem pomocí metody `runModelOnStreamFrames`. Výsledek detekce společně s časem a datem je následně cyklicky vypisován pod okno s otevřenou kamerou a zároveň do textového souboru. Tento textový soubor se nachází na výchozí cestě pro přístup k internímu úložišti mobilního telefonu. Konkrétně se jedná o umístění `Android\data\cz.vutbr.feec.utko.dp_object_detection_app\files`. Záznamy jsou ukládány ve vizuálním formátu, který je vidět na obrázku 4.3.



Obr. 4.3: Probíhající detekce objektů přes zadní kameru mobilního telefonu.

4.2 Model MobileNet v1

Prvním použitým a otestovaným modelem je model SSD MobileNet v1, který byl natrénován na datové množině COCO. Tento model obsahuje natrénované váhy s různými objekty, proto je použit pouze pro prvotní testování. Jeho velikost v optimalizované verzi je 27 MB a byl již otestován na mobilních zařízeních Google Pixel 3 (Android 10), Pixel 4 (Android 10) a iPhone XS (iOS 12.4.1) [55]. V rámci

vlastního testování je model použit na zařízení Samsung Galaxy A6+. Porovnání výsledků použitého modelu při detekci objektu je shrnuto v tabulce 4.2.

Tab. 4.1: Tabulka rychlosti detekce mobilních zařízení.

Mobilní zařízení	**CPU**	**GPU**	**RAM**	**Rychlost detekce [ms]**
Google Pixel 3	$4 \times 2,5$ GHz + $4 \times 1,6$ GHz	$0,71$ GHz	4 GB, $1,8$ GHz	22
Google Pixel 4	$1 \times 2,84$ GHz + $3 \times 2,42$ GHz + $4 \times 1,8$GHz	$0,67$ GHz	6 GB, $2,1$ GHz	20
iPhone XS	$2 \times 2,5$ GHz + $4 \times 1,59$ GHz	$4 \times 1,1$ GHz	4 GB	7,6
Samsung Galaxy A6+	$8 \times 1,8$ GHz	$0,65$ GHz	4 GB, $0,9$ GHz	156

Snímaným objektem je osobní automobil, který má v rámci natrénovaného modelu značku `"car"`. Pro záznam výsledků byl zvolen boční pohled na vozidlo. Použitý model obsahuje 80 detekovatelných kategorií mezi nimiž se z dopravních prostředků nachází automobil, jízdní kolo, motocykl, autobus, vlak, nákladní auto, loď a letadlo.

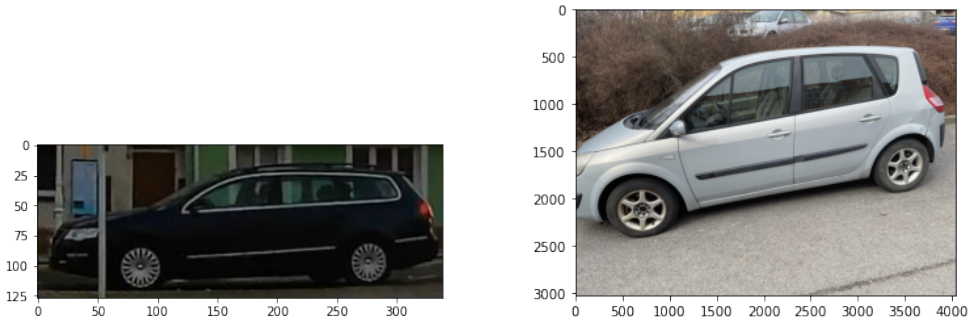
4.3 Vlastní model a vlastní datová množina

Pro trénování byla vytvořena vlastní, menší datová množina. S touto množinou jsou následně prováděny experimenty s vytvořenou konvoluční neuronovou sítí. Množina obsahuje celkem 622 obrazů, které byly vytvořeny pomocí fotoaparátu a snímků z palubní kamery automobilu. Obrazy jsou rozděleny do dvou tříd – automobil a dodávka. Snímky z fotoaparátu mají stejnou velikost, snímky z palubní kamery mají každý jinou velikost. Při vstupu do neuronové sítě tak bude nutné jejich předzpracování, kdy bude množina sjednocena. Data jsou pro trénování modelu rozdělena do tří podmnožin – trénovací, validační a testovací. Poměr rozdělení byl zvolen zhruba na: 70 % trénovací, 15 % validační a 15 % testovací. Ukázka vzorků z vytvořené datové množiny je na obrázku 4.4 a 4.5.

Pro experiment s vlastní datovou množinou je natrénován vlastní model. Konvoluční neuronová síť je postavena podle zpracované teorie a v průběhu trénování upravována tak, aby dosahovala co nejlepších výsledků. Vzhledem k tomu, že se sice jedná o kategorickou klasifikaci, ale třídy jsou pouze dvě, tak je nutné, aby model dosahoval daleko vyšší přesnosti než 50 %. Proto jsou v průběhu trénování na

Tab. 4.2: Počet vzorků a rozdělení vytvořené množiny.

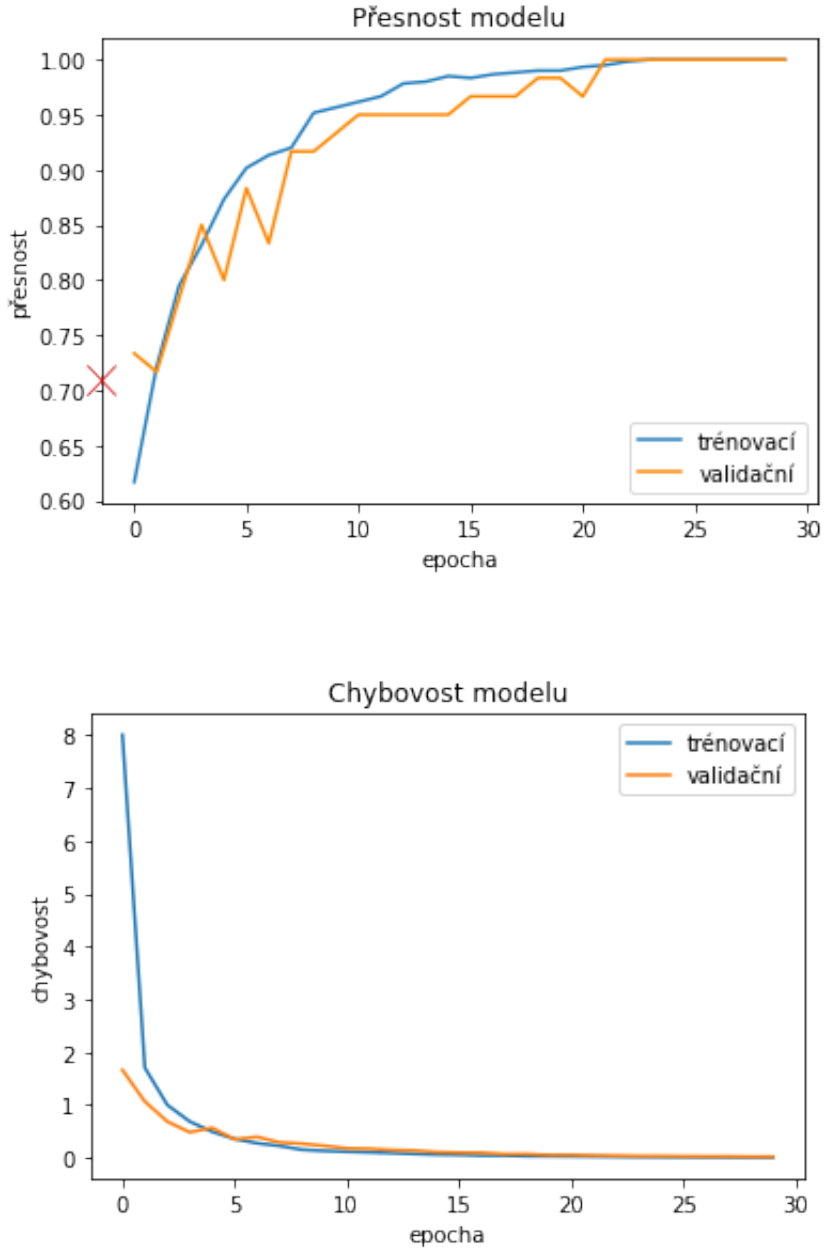
Třída	**Počet vzorků množiny**		
	Trénovací	Validační	Testovací
Automobil	260	51	51
Dodávka	260	51	51
Celkem	622		



Obr. 4.4: Ukázka vzorků třídy „auto“.

Obr. 4.5: Ukázka vzorků třídy „dodávka“.

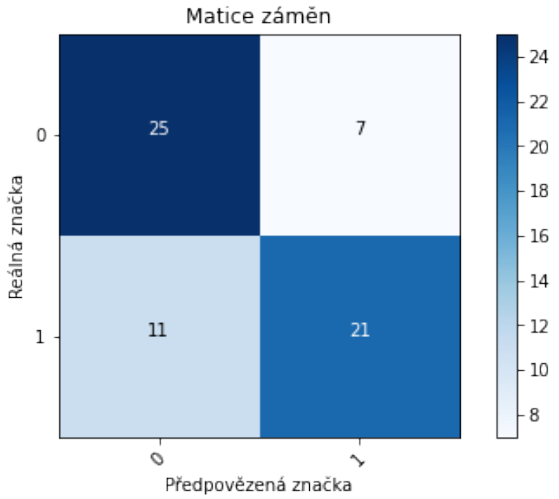
základě validační a testovací přesnosti upravovány parametry sítě (použití různých vrstev, počet neuronů, počet a kombinace vrstev) tak, aby bylo dosaženo co nejlepší testovací přesnosti. Celkem bylo spuštěno přes 50 trénovacích cyklů a konfigurací, kdy nejlepších výsledků na dané datové množině dosáhla neuronová síť 17 [1]. Průběh trénování je vynesen do grafů 4.6, kde je vidět validační přesnost, chybovost modelu při trénování a křížkem označena závěrečná testovací přesnost při vyhodnocování na sítí dosud „neviděných“ datech.



Obr. 4.6: Průběh trénování vlastního modelu.

[1]Pole otazníku v závorce udává nespecifikovaný parametr dávkování dat.

Při trénování vlastního modelu bylo vyzkoušeno mnoho konfigurací sítě. Architektura 17 vykazovala na testovacích datech nejlepších výsledků přesnosti. Z grafu validační a trénovací přesnosti je vidět, že během 30 epoch dosáhly svého maxima, což vzhledem k dosažené testovací přesnosti značí mírnou přetrénovanost. Model byl schopen bezchybně určovat, do které třídy data patří, ale pouze na těch datech, která už byla natrénována. Na datech, která nebyla natrénována pak správně přiřadil do patřičné třídy s celkovou přesností $71,88$ %. Pro podrobnější analýzu toho, co se daný model naučil klasifikovat nejlépe a nejhůře je možné si vygenerovat tzv. matici záměn (z anglického *confusion matrix*). Tato matice je vhodným nástrojem pro vylepšování klasifikace modelu. Je z ní možné vyčíst, která třída byla natrénována nejlépe, nebo která naopak nejhůře. Z tohoto důvodu je používána pro vhodnější vyvážení dat nebo k přesnějším úpravám struktury neuronové sítě. Pro přehlednější vyobrazení jsou v matici jednotlivé třídy zastoupeny číslem (značka). Výsledná matice záměn je zobrazena na obrázku 4.7. Značka 0 v tomto případě zastupuje třídu auto, značka 1 dodávku. Je vidět, že se model naučil o něco lépe klasifikovat auto než dodávku. Z 32 případů správně určil třídu auto ve 25 případech, dodávku pak ve 21 případech a 11× ji zaměnil s autem. Vzhledem k omezené datové množině, kdy některé obrazy obsahovaly velké množství šumu je možné tento výsledek považovat za uspokojující.



Obr. 4.7: Matice záměn prvního modelu binární klasifikace.

4.4 Vlastní model s rozšířením vlastních dat

Při trénování vlastního modelu neuronové sítě byla vyzkoušena také technika rozšiřování dat. Tato technika spočívá v náhodných transformacích obrazu trénovací

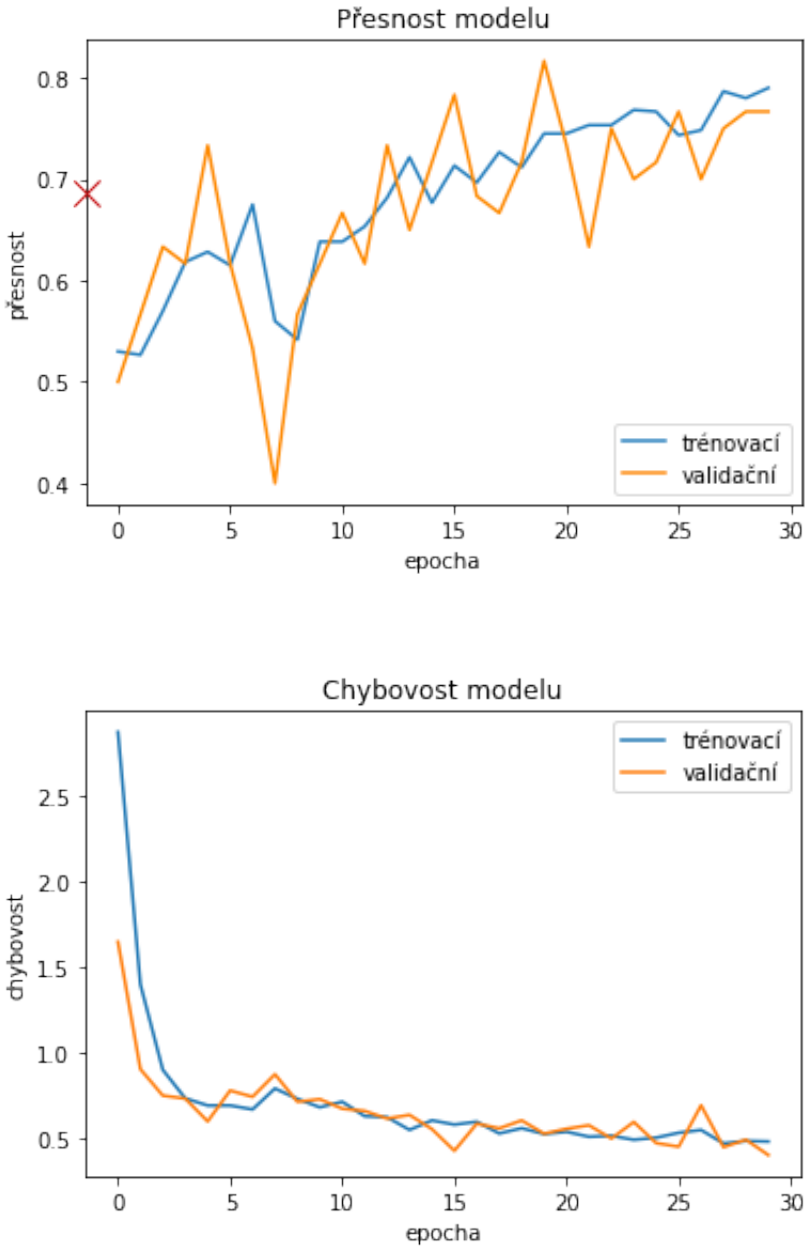
datové množiny. Tyto transformované obrazy jsou posílány do sítě, která má tímto možnost trénovat na variabilnějších datech. V každé trénovací epoše je aplikována transformace dle zadaných parametrů jako například náhodná rotace, horizontální a vertikální posun či převrácení, prostorové zkreslení nebo přiblížení. Obrazy ale nejsou jiné než ty, které jsou reálně k dispozici. Jedná se sice o techniku navýšení dat, ale jejich počet se nemění. Jsou pouze aplikovány transformace, které mají za cíl poskytnout základ pro robustnější a méně přetrénovaý model.
Na vlastní datovou množinu bylo aplikováno rozšíření dat. Vzorky jsou pro ukázku zobrazeny na obrázku 4.8. Trénovací data nyní obsahují rozdíly, které přesně určují tyto transformace [56]:

1. **Rotace** – Data jsou náhodně otočena o jednotky stupňů v zadaném rozmezí.
2. **Horizontální a vertikální posun** - Data jsou náhodně posunuta v daném směru o jednotky datového typu `float`.
3. **Prostorové zkreslení** – Data jsou prostorově zkreslena ve směru proti hodinovým ručičkám ve stupních. Parametr je zadáván v datovém typu `float`.
4. **Přiblížení** – Tento parametr je možné zadat jako rozsah nebo také v datovém typu `float`. Ve výchozím stavu určuje rozsah přiblížení obrazu do středu.
5. **Horizontální a vertikální zrcadlení** – Logický parametr `true` nebo `false`, který zajišťuje zrcadlení obrazu podle zadané osy.

Při technice navyšování dat je třeba dbát na relevanci trénovacích dat dodávaných do sítě. Z tohoto důvodu nemohou být použity některé transformace jako například vertikální zrcadlení. Vzhledem k tomu, že dopravní prostředky nejsou většinou snímány v rotaci o 180°, není třeba síť na zrcadlená data trénovat. Takto transformovaná data nejsou v testovací datové množině a zbytečně by tak přinesla nepřesnosti do modelu. Při některých transformacích také vzniká na datech volné místo. V tomto případě je nastaveno, že se toto místo vyplní barvou posledního okrajového pixelu.

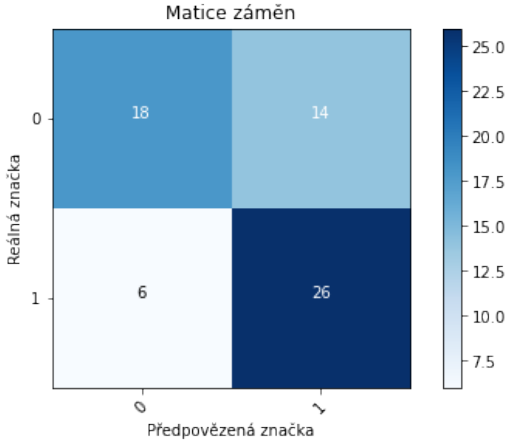
Obr. 4.8: Ukázka vygenerovaného a transformovaného obrazu.

Přes velmi kolísavý průběh trénování a testování na validačních datech model dosáhl přesnosti 68, 75 %. Struktura sítě byla mnohokrát upravována tak, aby průběh validování byl co nejstálejší. Síť už nevykazuje takovou přetrénovanost jako 17. Matice záměn 4.10 ukazuje, že model je naopak od prvního experimentu více naučen rozpoznávat dodávky, které správně určil 26× z 32 případů. Automobil správně



Obr. 4.9: Průběh trénování vlastního modelu s rozšířením dat.

určil 18×. Vzhledem k malému počtu dat a podobnosti obou tříd může mít určování spíše náhodný charakter, protože dosažená přesnost 68, 75 % tvoří od náhodného určování dvou tříd (50 %) malý rozdíl.



Obr. 4.10: Matice záměn modelu binární klasifikace s rozšířením dat.

4.5 Vlastní model s dostupnou datovou množinou vozidel

Dalším vyzkoušeným přístupem k natrénování modelu na rozpoznávání vozidel je vytvoření vlastní neuronové sítě a její natrénování na volně dostupné datové množině. Pro tento experiment byla použita datová množina s názvem *TAU Vehicle Type Recognition Competition*. Datová množina je součástí soutěže, které se může zúčastnit každý se zájmem o konvoluční neuronové sítě a pomocí různých technik dosáhnout na této datové množině co nejvyšší přesnosti. Nejvyšší dosaženou přesností je zatím 92, 94 %. Surová data obsahují 17 tříd typů vozidel rozdělených do trénovací a testovací množiny [57].

Pro tento experiment byla data upravena. Počet vzorků v třídách byl vyvážen na stejný počet ve všech množinách. Data byla rozdělena v poměru 70 % trénovací, 15 % validační a 15 % testovací množina. Také bylo použito pouze 6 tříd z dostupných 17. Těmito třídami je automobil, autobus, dodávka, jízdní kolo, motocykl a nákladní automobil. Experimentálně bylo dosaženo přesnosti 59, 89 % na testovacích datech se strukturou sítě 19. Průběhy trénování s vyznačenou testovací přesností jsou vidět na grafech 4.12.

Výslednou přesnost 59, 89 % je možné považovat za dostačující. Trénování probíhá na omezeném hardwaru bez jakéhokoliv předzpracování dat. Vzhledem k tomu, že je model trénován na 6 třídách označených jako N, je podle rovnice 4.1 mezní přesnost

Acc_m pro začátek trénování 16, 67 %.

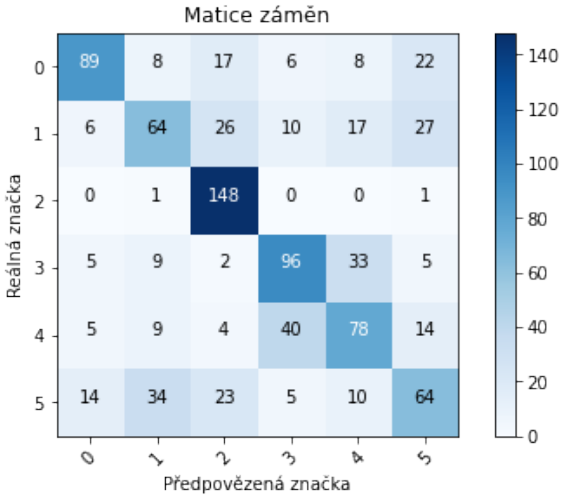
$$Acc_m = \frac{100}{N} \quad [\%] \tag{4.1}$$

S touto přesností model přiřazuje náhodně data do tříd. Cílem je tuto přesnost co nejvíce zvýšit a zvětšit tak poměr klasifikování pro danou třídu. Matice záměn 4.11 značí, že nejlépe byla klasifikována třída se značkou 2 (dodávka). Značky odpovídají následnému přiřazení tříd:

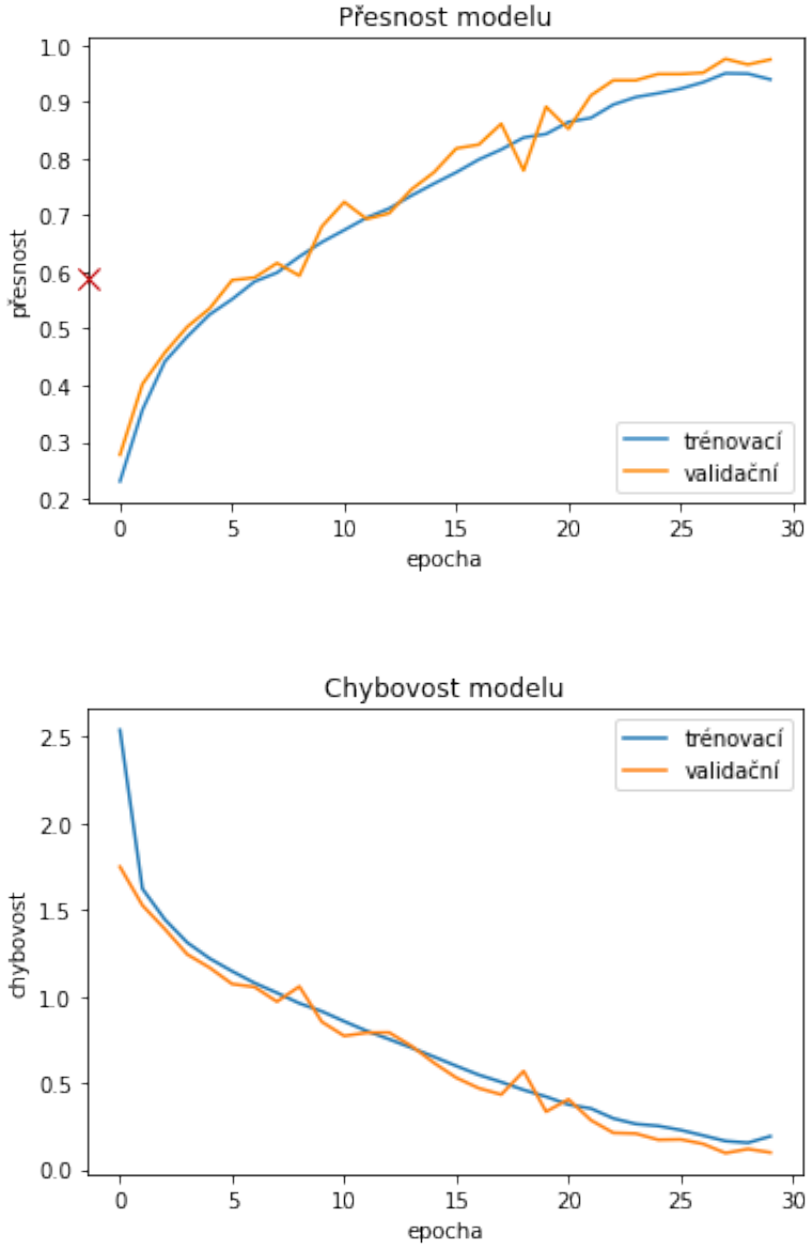
```
In[]    test_generator.class_indices
Out[]   {'autobus': 0,
         'automobil': 1,
         'dodavka': 2,
         'jizdni_kolo': 3,
         'motocykl': 4,
         'nakladni_auto': 5}
```

Výpis 4.2: Přiřazení tříd ke značkám.

Naopak nejhůře jsou natrénovány třídy klasického a nákladního automobilu. Při podrobnějším zkoumání dat bylo zjištěno, že datová množina tyto třídy často zaměňuje. Třídy automobilu, dodávky a nákladního automobilu jsou v datech často vnímány jinak než v České Republice. Datová množina je poskládaná z obrazů vozidel z celého světa a vozidlo, které je v Česku vnímáno například jako dodávka se nachází v třídě nákladního auta a naopak. Taktéž se jedná o třídu automobilu, která se v datech nachází ve třídě dodávky a naopak. Je pak pochopitelné, že tyto tři třídy jsou při předpovědi často zaměněny. Při trénování je tak třeba dbát na to, aby byly tyto rozdíly správně použity pro danou aplikaci. U klasifikátorů v jiných zemích je pro vyšší přesnost nutné, aby jejich natrénování proběhlo na lokálních datech.



Obr. 4.11: Matice záměn modelu 19.



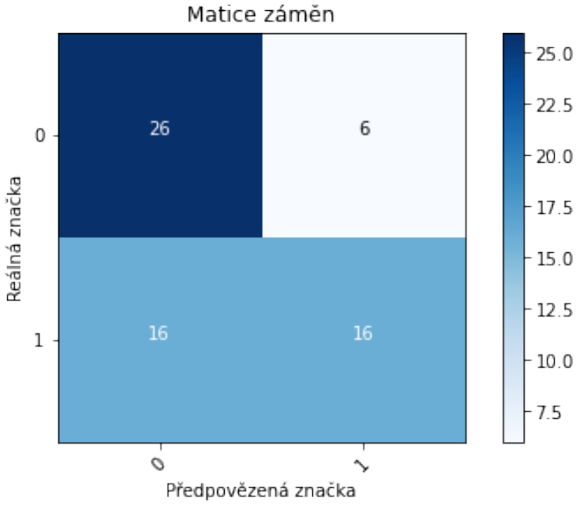
Obr. 4.12: Průběh trénování modelu na datové množině vozidel.

4.6 Předtrénovaný model s dotrénováním o vlastní data

Posledním přístupem pro dosažení cíle klasifikace vozidel je použití již natrénovaného modelu. Tento model je pomocí přeneseného učení dotrénován o vlastní vytvořená data. K tomuto účelu je použit model MobileNet, ale je možné použít kterýkoliv dostupný model. MobileNet obsahuje váhy datové množiny ImageNet a byl vybrán z důvodu jeho nenáročnosti a menší velikosti (17 MB).

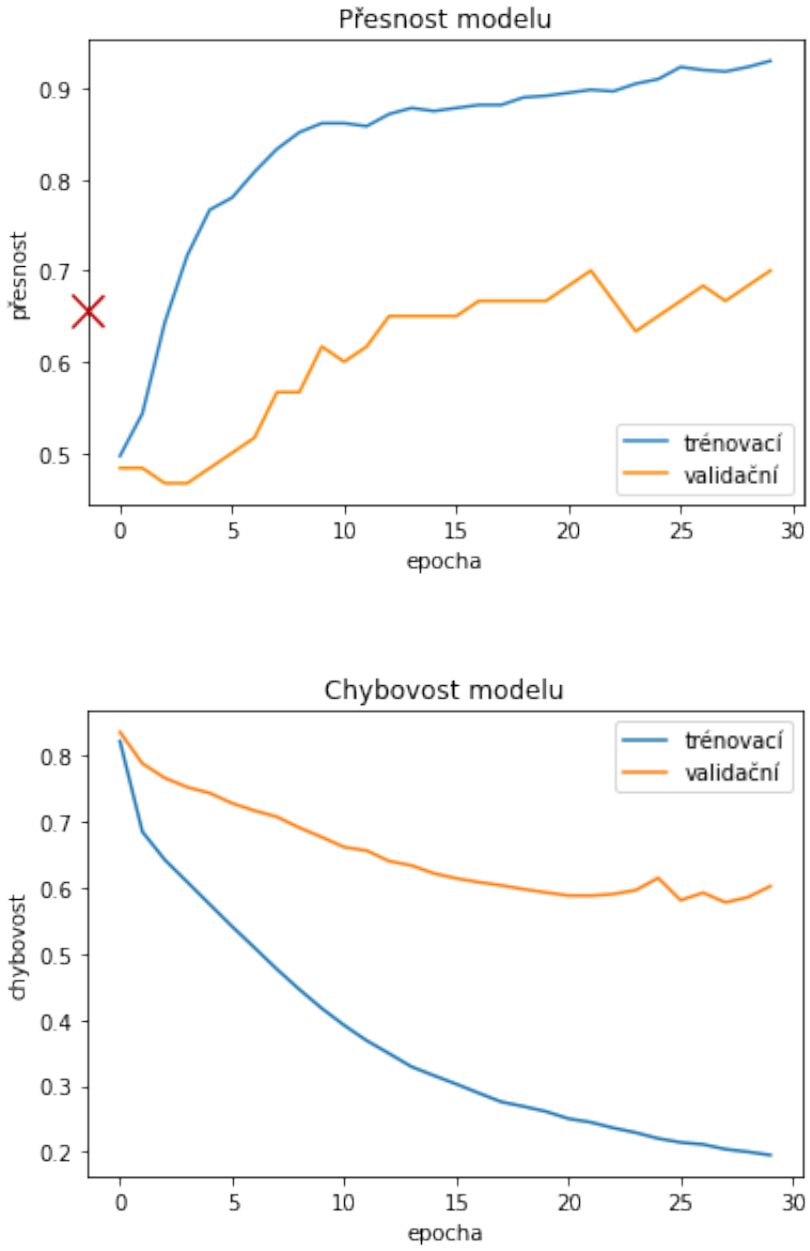
Pro dotrénování vah modelu je nutné „uzamknout“ jeho vrstvy kromě poslední, která přiřazuje data do správných tříd. V tomto případě se jedná o vrstvu s 1000 neurony, protože MobileNet je natrénován na 1000 třídách. Po jejich zmrazení jsou přidány jiné konečné vrstvy, které po natrénování obsahují natrénované váhy vlastních dat. Klasifikace probíhá s využitím naučených příznaků z předtrénovaného modelu, ale klasifikuje pouze dotrénované třídy. Jako první byly dotrénovány pouze dvě třídy z vlastní datové množiny – auto a dodávka. K základní architektuře MobileNet bylo přidáno 5 vrstev, které jsou vidět na obrázku struktury 20.

Průběh dotrénování s testovací přesností je vynesen do grafů 4.14. Vlastní datová množina byla mírně upravena ořezáním některých obrazů, na kterých bylo příliš mnoho přebytečného šumu, které přinášely nepřesnosti do sítě. Při dotrénování byly použity převážně `Dense` vrstvy s větším počtem jednotek. Tímto způsobem bylo dosaženo konečné testovací přesnosti $65,62$ %. Matice záměn 4.13 značí, že třída dodávky není vhodně natrénovaná, protože vykazuje charakter náhodného určování.



Obr. 4.13: Matice záměn dotrénovaného modelu – binární klasifikace.

Druhým a zároveň finálním experimentem je dotrénování předtrénovaného modelu o 6 kategorií vozidel. Těmito kategoriemi jsou vozidla, která se vyskytují na



Obr. 4.14: Průběh dotrénování modelu na vlastní datové množině.

silnicích nejčastěji. Patří mezi ně autobus, automobil, dodávka, jízdní kolo, motocykl a nákladní auto. V rámci tohoto experimentu bylo navíc do datové množiny dodáno 43 vzorků nákladních aut. Data jsou rozdělena v poměru 70 % trénovací, 15 % validační a 15 % testovací množina, kdy trénovací množinu automobilů, dodávek a nákladních aut tvoří vlastní data, která jsou doplněna o data z datové množiny *TAU Vehicle Dataset*. Počet trénovacích dat každé z šesti tříd činí 700 obrazů, validační data obsahují 150 obrazů vybraných z trénovací množiny, stejně tak testovacích, které jsou nezávislé. Struktura sítě, postavené nad modelem MobileNet (v obrázku označeném jako „Model“) je na obrázku 21. Aktivační funkcí poslední vrstvy je `softmax` a dotrénování sítě optimalizuje optimizátor `Adam` s koeficientem učení 1×10^{-4}. Průběh učení s vyznačenou testovací přesností je vyznačen na grafech 4.15.

Tab. 4.3: Rozdělení a počet vzorků datové množiny.

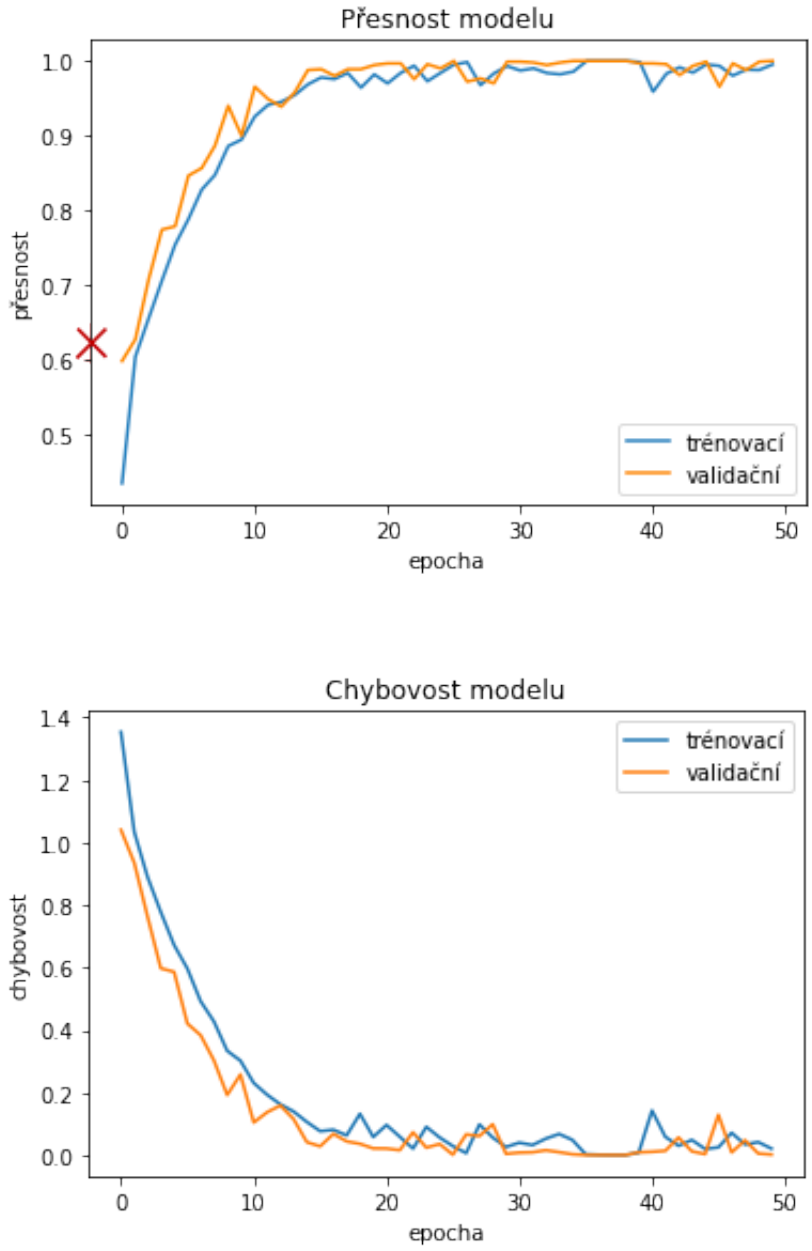
Třída	**Počet vzorků množiny**		
	Trénovací	Validační	Testovací
Autobus	700	150	150
Automobil	700	150	150
Dodávka	700	150	150
Jízdní kolo	700	150	150
Motocykl	700	150	150
Nákladní auto	700	150	150
Celkem	5100		

Dotrénovaný model dosáhl klasifikační přesnosti 62, 33 %. Každá z 6 tříd obrazů tak bude správně přiřazena do správné kategorie s touto pravděpodobností. Přiřazení značek odpovídá stejnému rozdělení, jako v předešlém experimentu:

```
{'autobus': 0,
'automobil': 1,
'dodavka': 2,
'jizdni_kolo': 3,
'motocykl': 4,
'nakladni_auto': 5}
```

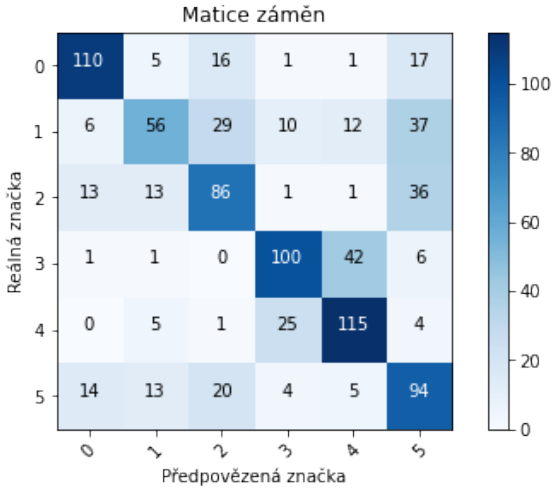
Výpis 4.3: Přiřazení tříd ke značkám.

Z matice 4.16 je vidět, že třída motocyklu byla určována s největší úspěšností. Z celkem 150 testovacích vzorků jich správně bylo určeno 115. Nejvíce model zaměňoval motocykl s jízdním kolem, kdy určil, že 25 motocyklů je jízdní kolo. Nejhorší



Obr. 4.15: Průběh dotrénování modelu na vlastní datové množině.

výsledky klasifikace má klasický automobil. Správně bylo určeno 56 klasických automobilů ze 150, kdy nejvíce model zaměňoval automobil s nákladním autem a dodávkou.



Obr. 4.16: Matice záměn dotrénovaného modelu.

4.7 Porovnání modelů a využití hardwaru

U všech výše zmíněných modelů byly kromě přesnosti a podrobnějších dat o klasifikační schopnosti modelu zaznamenány také údaje o využití hardwaru. V tabulce 4.7 je uvedeno 5 použitých modelů a jejich velikost ihned po natrénovaní. Tento soubor odpovídá souboru s příponou `.h5` a obsahuje vícerozměrná pole numerických dat, která se řídí danou hierarchií [58]. Dále je model převeden do „odlehčené“ varianty s příponou `.tflite` vhodné k použití na omezeném hardwaru. Následně je přes aplikaci spuštěna klasifikace a přes vývojové prostředí Android Studio a externí monitorovací aplikaci *Simple System Monitor* jsou zaznamenány průměrné hodnoty využití GPU a RAM. Je sledována rychlost detekce modelu a zaznamenána její průměrná hodnota. Tento test je spuštěn pro každý model po stejnou dobu 30 vteřin.

Tabulka 4.7 shrnuje měřené parametry modelů v závislosti na hardwaru mobilního zařízení. Čísla modelů odpovídají provedeným experimentům:

- Model 1 – vlastní model a vlastní datová množina (klasifikace 2 tříd)
- Model 2 – vlastní model s rozšířením vlastních dat (klasifikace 2 tříd)
- Model 3 – vlastní model s dostupnou datovou množinou (klasifikace 6 tříd)
- Model 4 – předtrénovaný model s dotrénováním o vlastní data (klasifikace 2 tříd)
- Model 5 – předtrénovaný model s dotrénováním o vlastní data (klasifikace 6 tříd)

Tab. 4.4: Shrnutí modelů a využití hardwaru.

Samsung Galaxy A6+						
Model	Velikost původního modelu	Velikost modelu po převedení do .tflite	Využití GPU [%]	Využití RAM [%]	Přesnost modelu [%]	Rychlost detekce [ms]
1	42,6 kB	20,7 kB	11,1	5,9	71,88	160
2	178 kB	46,1 kB	12,8	5,1	68,75	157
3	0,98 MB	984 kB	12,5	7	59,89	158
4	12,9 MB	12,3 MB	24,2	6	65,62	550
5	43,6 MB	22,5 MB	25,6	5	62,33	556

Sloupce uvádějící velikosti modelů před a po jejich kompresi do formátu `.tflite` jsou uvedeny pro porovnání změny velikosti modelu pro mobilní zařízení. Tento parametr není možné přímo ovlivnit a je dán architekturou Tensorflow Lite. Při testování na mobilním telefonu byl větší rozdíl ve využití GPU zaznamenán při použití předtrénovaného modelu, který byl následně dotrénován. Počet dotrénovaných tříd neměl na využití GPU, RAM a rychlost detekce velký vliv. Při přechodu z natrénování vlastního modelu bez jakýchkoliv předtrénovaných parametrů na předtrénovaný model MobileNet s dotrénováním o vlastní data byl zjištěn dvojnásobný rozdíl ve využití GPU a téměř 4 násobný rozdíl v rychlosti detekce. Pro běžného uživatele je rychlost detekce v milisekundách těžko postřehnutelná, ale pro využití v dopravě může být tento rozdíl nevyhovující. Rozdíl je dán zpožděním sítě, kdy sice podává jistější výsledky klasifikace díky předtrénovaným příznakům, ale zároveň trvá déle, než daný objekt klasifikuje právě kvůli množství příznaků. Z tohoto zjištění vyplývá, že v praktických případech statických dopravních kamer s omezeným hardwarem je pro řešení citlivá na rychlost klasifikace vhodnější natrénování neuronové sítě bez předtrénovaných vah. Je ale třeba mít k dispozici dostatek maximálně optimalizovaných dat a experimentálně zvyšovat přesnost klasifikace, což je nelehký a zdlouhavý proces.

Závěr

Tato práce se zabývá klasifikací pohybujících se objektů ve videu s využitím neuronových sítí pomocí Android aplikace. V teoretické části byl zpracován úvod do strojového učení a současný stav vědy a techniky v oblasti konvolučních neuronových sítí pro zpracování obrazu. Byly popsány mechanismy, kterými se neuronové sítě trénují a optimalizují, techniky trénování a datové množiny.

V experimentální části byly zmíněny a popsány využité aplikační rozhraní a stavební bloky pro tvoření a trénování konvolučních neuronových sítí. V rámci této části byla vytvořena mobilní aplikace určená pro platformu Android, která dokáže v reálném čase klasifikovat účastníky dopravního provozu a ukládat časové značky klasifikace. Jako první byla funkčnost aplikace otestována na předtrénovaném modelu SSD MobileNet v1, kdy byla zaznamenána rychlost klasifikace na zařízení Samsung Galaxy A6+. Následovala série pěti experimentů.

Pro tyto experimenty byla vytvořena vlastní datová množina. Tato množina se skládá ze dvou tříd a obsahuje celkem 622 obrazů automobilu, dodávky a také 43 nákladních aut. Jako první byla datová množina použita pro trénování vlastního modelu, poté byla rozšířena pomocí techniky navyšování dat a model byl znovu natrénován. Třetím experimentem bylo natrénování vlastní neuronové sítě na dostupné datové množině vozidel. V tomto případě se experiment rozdělil nejprve na binární klasifikaci a poté na více-třídní klasifikaci. Závěrečným experimentem bylo dotrénování hojně používaného modelu MobileNet o vlastní data šesti tříd. Tato data se skládala z vlastních obrazů a také z dat dostupné množiny vozidel. Následně bylo porovnáno využití hardwaru mobilního zařízení pro všechny natrénované modely.

Hlavním přínosem práce je vytvoření klasifikační aplikace se záznamem výsledků detekce do externího souboru, otestování modelů v závislosti na omezeném hardwaru a zjemnění kategorií vozidel. Mnoho tříd v datech totiž obsahuje obrazy vozidel, které mohou být v různých zemích světa vnímány jinak. Často tak dochází k zaměňování dat a nepřesné klasifikaci. Pro použití v České Republice může vytvořená datová množina do jisté míry tento problém minimalizovat. Vytvořená Android aplikace je připravena pro použití v reálném provozu. Vzhledem ke zvolenému přístupu k jejímu vývoji je také možná rychlá migrace při potřebě použití aplikace na operační systém iOS.

Literatura

[1] *A Perspective of the Noise Removal for Faster Neural Network Training* [online]. M. Rajnoha, V. Mikulec, R. Burget and J. Drazil, 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Dublin, Ireland, 2019, pp. 1-4, doi: 10.1109/ICUMT48472.2019.8970907 [cit. 19. 10. 2020]. Dostupné z URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8970907>.

[2] *Simple Convolutional Neural Network on Image Classification* [online]. Tianmei Guo, Jiwen Dong, Henjian Li´Yunxing Gao [cit. 19. 10. 2020]. Dostupné z URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8078730>.

[3] *Towards robust voice pathology detection* [online]. Harar, P., Galaz, Z., Alonso-Hernandez, J.B. et al. Neural Comput & Applic 32, 15747–15757 (2020). [cit. 26. 10. 2020]. Dostupné z URL: <https://doi.org/10.1007/s00521-018-3464-7>.

[4] *Introduction to machine learning.*. ALPAYDIN, Ethem. MIT press, 2020. [cit. 26. 10. 2020].

[5] *Supervised machine learning: A review of classification techniques.* [online]. KOTSIANTIS, Sotiris B.; ZAHARAKIS, I.; PINTELAS, P. Emerging artificial intelligence applications in computer engineering, 2007, 160.1: 3-24. [cit. 30. 10. 2020]. Dostupné z URL: <http://www.informatica.si/index.php/informatica/article/viewFile/148/140>.

[6] *A survey of Apple A11 Bionic processor.* [online]. Sarkar, Subham. (2018). [cit. 30. 10. 2020]. Dostupné z URL: <https://www.researchgate.net/publication/340580371_A_survey_of_Apple_A11_Bionic_processor>.

[7] *Etudy o mozku a myšlení*, Praha: Vysoká škola ekonomická, 1994. ISBN 80-7079-280-9. PSTRUŽINA, Karel. [cit. 1. 11. 2020].

[8] *A Logical Calculus of the Ideas Immanent in Nervous Activity.* [online]. W. S. McCulloch, W. H. Pitts. [cit. 1. 11. 2020]. Dostupné z URL: <http://aiplaybook.a16z.com/reference-material/mcculloch-pitts-1943-neural-networks.pdf>.

[9] *Neural networks* [online]. Ch. Stergiou, D. Siganos. [cit. 1. 11. 2020]. Dostupné z URL: <http://srii.sou.edu.ge/neural-networks.pdf>.

[10] *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain* [online]. F. ROSENBLATT, Cornell Aeronautical Laboratory [cit. 1. 11. 2020]. Dostupné z URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf>.

[11] *Parameter optimization in neural networks* [online]. Katanforoosh, Kunin et al., deeplearning.ai, 2019. [cit. 2. 11. 2020]. Dostupné z URL: <https://www.deeplearning.ai/ai-notes/optimization/>.

[12] *The Gradient and Directional Derivative* [online]. 1996 Department of Mathematics, Oregon State University [cit. 2. 11. 2020]. Dostupné z URL: <http://sites.science.oregonstate.edu/math/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/grad/grad.html>.

[13] *Gradient Descent Visualization* [online]. simar (2020), MATLAB Central File Exchange. Retrieved November 3, 2020. [cit. 3. 11. 2020]. Dostupné z URL: <https://www.mathworks.com/matlabcentral/fileexchange/35389-gradient-descent-visualization>.

[14] *A Comparative Analysis of Gradient Descent-Based Optimization Algorithms on Convolutional Neural Networks* [online]. E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala, C. O. Aigbavboa [cit. 3. 11. 2020]. Dostupné z URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8769211>.

[15] *Train faster, generalize better: Stability of stochastic gradient descent* [online]. Moritz Hardt, Benjamin Recht, Yoram Singer. [cit. 3. 11. 2020]. Dostupné z URL: <http://proceedings.mlr.press/v48/hardt16.pdf>.

[16] *Lecture 9: Generalization* [online]. Roger Grosse [cit. 3. 11. 2020]. Dostupné z URL: <https://www.cs.toronto.edu/~lczhang/321/notes/notes09.pdf>.

[17] *Optimization for machine learning. Cambridge (Mass.): The MIT Press.* [online]. Wright, S. J., Sra, S., &; Nowozin, S. (2012). [cit. 3. 11. 2020]. Dostupné z URL: <https://www.google.com/books/edition/_/JPQx7s2L1A8C?hl=en&gbpv=1&pg=PA351>.

[18] *Lecture 6.5 - rmsprop, COURSERA: Neural Networks for Machine Learning* [online]. Tieleman, T. and Hinton, G. (2012) [cit. 4. 11. 2020]. Dostupné z URL: <https://climin.readthedocs.io/en/latest/rmsprop.html>.

[19] *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION* [online]. Diederik P. Kingma, Jimmy Lei Ba. [cit. 4. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1412.6980.pdf>.

[20] *Weight (Artificial Neural Network)* [online]. [cit. 9. 11. 2020]. Dostupné z URL: <https://deepai.org/machine-learning-glossary-and-terms/weight-artificial-neural-network>.

[21] *Activation functions in neural networks.* [online]. SHARMA, Sagar. Towards Data Science, 2017, 6. [cit. 7. 11. 2020]. Dostupné z URL: <https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf>.

[22] *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning* [online]. Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall [cit. 7. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1811.03378.pdf>.

[23] *Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study)* [online]. Haider Khalaf Jabbar, Dr. Rafiqul Zaman Khan [cit. 10. 11. 2020]. Dostupné z URL: Odkaz na zdroj.

[24] *Reccurent Neural Networks* [online]. L. R. Medsker, L. C. Jain [cit. 10. 11. 2020]. Dostupné z URL:
Odkaz na zdroj.

[25] *Vision-based vehicle detection and counting system using deep learning in highway scenes. Eur. Transp. Res. Rev. 11, 51 (2019).* [online]. Song, H., Liang, H., Li, H. et al. [cit. 24. 10. 2020]. Dostupné z URL: <https://doi.org/10.1186/s12544-019-0390-4>.

[26] *Vehicle type detection based on deep learning in traffic scene* [online]. [cit. 24. 10. 2020]. Dostupné z URL: <https://doi.org/10.1016/j.procs.2018.04.281>.

[27] *Live Target Detection with Deep Learning Neural Network and Unmanned Aerial Vehicle on Android Mobile Device* [online]. [cit. 24. 10. 2020]. Dostupné z URL: <http://indexive.com/uploads/papers/icatces2018-158.pdf>.

[28] *CS231n Convolutional Neural Networks for Visual Recognition* [online]. [cit. 13. 11. 2020]. Dostupné z URL: <https://cs231n.github.io/classification/>.

[29] *Vliv pozadí a velikosti databáze na trénování neuronových sítí pro klasifikaci obrazů* [online]. MIKULEC, Vojtěch. [online]. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Martin Rajnoha. [cit. 14. 11. 2020].

Dostupné z URL: <https://www.vutbr.cz/studenti/zav-prace/detail/118072>.

[30] *Gabor filter for image processing and computer vision* [online]. N. Petkov and M.B. Wieling, University of Groningen, Department of Computing Science, Intelligent Systems [cit. 14. 11. 2020]. Dostupné z URL: <http://matlabserver.cs.rug.nl/edgedetectionweb/web/edgedetection_params.html>.

[31] *NEURAL NETWORK-BASED SEGMENTATION OF TEXTURES USING GABOR FEATURES* [online]. G. Ramakrishnan, S. Kumar Raja, H. V. Raghu [cit. 14. 11. 2020]. Dostupné z URL: <http://eprints.iisc.ac.in/5152/1/neural_networks.pdf>.

[32] *CS231n Convolutional Neural Networks for Visual Recognition – CNN* [online]. [cit. 15. 11. 2020]. Dostupné z URL: <https://cs231n.github.io/convolutional-networks/>.

[33] *CS231n Convolutional Neural Networks for Visual Recognition – Transfer Learning* [online]. [cit. 16. 11. 2020]. Dostupné z URL: <https://cs231n.github.io/transfer-learning/>.

[34] *YOLOv3: An Incremental Improvement* [online]. Joseph Redmon, Ali Farhadi University of Washington [cit. 16. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1804.02767.pdf>.

[35] *YOLOv4: Optimal Speed and Accuracy of Object Detection* [online]. Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao [cit. 16. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/2004.10934.pdf>.

[36] *YOLOv5 New Version - Improvements And Evaluation* [online]. [cit. 18. 11. 2020]. Dostupné z URL: <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>.

[37] *Data augmentation* [online]. [cit. 19. 11. 2020]. Dostupné z URL: <https://www.tensorflow.org/tutorials/images/data_augmentation>.

[38] *COCO Object Detection* [online]. [cit. 19. 11. 2020]. Dostupné z URL: <https://benchmarks.ai/coco-detection>.

[39] *EfficientDet: Scalable and Efficient Object Detection* [online]. Mingxing Tan, Ruoming Pang, Quoc V. Le Google Research, Brain Team [cit. 19. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1911.09070.pdf>.

[40] *AN IMAGE IS WORTH 16X16 WORDS:TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE* [online]. Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby, Google Research, Brain Team [cit. 20. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/2010.11929v1.pdf>.

[41] *THE MNIST DATABASE of handwritten digits* [online]. Yann LeCun, Courant Institute, NYU Corinna Cortes, Google Labs, New York Christopher J.C. Burges, Microsoft Research, Redmond [cit. 20. 11. 2020]. Dostupné z URL: <http://yann.lecun.com/exdb/mnist/>.

[42] *Regularization of Neural Networks using DropConnect* [online]. Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus [cit. 20. 11. 2020]. Dostupné z URL: <http://yann.lecun.com/exdb/publis/pdf/wan-icml-13.pdf>.

[43] *Big Transfer (BiT): General Visual Representation Learning* [online]. [cit. 20. 11. 2020]. Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby Google Research, Brain Team Dostupné z URL: <https://arxiv.org/pdf/1912.11370.pdf>.

[44] *The CIFAR-10 dataset* [online]. Alex Krizhevsky, 2009 [cit. 20. 11. 2020]. Dostupné z URL: <https://www.cs.toronto.edu/~kriz/cifar.html>.

[45] *Detection and Classification of Vehicles* [online]. Surendra Gupte, Osama Masoud, Robert F. K. Martin and Nikolaos P. Papanikolopoulos [cit. 20. 11. 2020]. Dostupné z URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=994794>.

[46] *MSI GeForce RTX 2070 Armor 8GB* [online]. [cit. 22. 11. 2020]. Dostupné z URL: <https://asset.msi.com/pdf/main/global/presale/GeForce-RTX-2070-ARMOR-8G>.

[47] *Comparison of Deep Learning in Neural Networks on CPU and GPU-based frameworks* [online]. Kamil Aida-zade1, Elshan Mustafayev2, Samir Rustamov3 [cit. 22. 11. 2020]. Dostupné z URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8687085>.

[48] *Tensorflow* [online]. [cit. 21. 11. 2020]. Dostupné z URL: <https://www.tensorflow.org/>.

[49] *Tensorflow Lite* [online]. [cit. 21. 11. 2020]. Dostupné z URL: <https://www.tensorflow.org/lite>.

[50] *Keras* [online]. [cit. 21. 11. 2020]. Dostupné z URL: <https://keras.io/about/>.

[51] *Android Studio* [online]. [cit. 18. 3. 2021]. Dostupné z URL: <https://developer.android.com/studio/intro>.

[52] *Flutter FAQ* [online]. [cit. 20. 3. 2021]. Dostupné z URL: <https://flutter.dev/docs/resources/faq>.

[53] *Jupyter Notebook* [online]. [cit. 18. 3. 2021]. Dostupné z URL: <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>.

[54] *Keras layers API* [online]. [cit. 22. 11. 2020]. Dostupné z URL: <https://keras.io/api/layers/>.

[55] *Tensorflow - Object Detection* [online]. [cit. 25. 11. 2020]. Dostupné z URL: <https://www.tensorflow.org/lite/models/object_detection/overview>.

[56] *Image data preprocessing* [online]. [cit. 21. 4. 2021]. Dostupné z URL: <https://keras.io/api/preprocessing/image/>.

[57] *TAU Vehicle Type Recognition Competition* [online]. [cit. 26. 4. 2021]. Dostupné z URL: <https://www.kaggle.com/c/vehicle/overview>.

[58] *.H5File Extension* [online]. [cit. 6. 5. 2021]. Dostupné z URL: <https://fileinfo.com/extension/h5>.

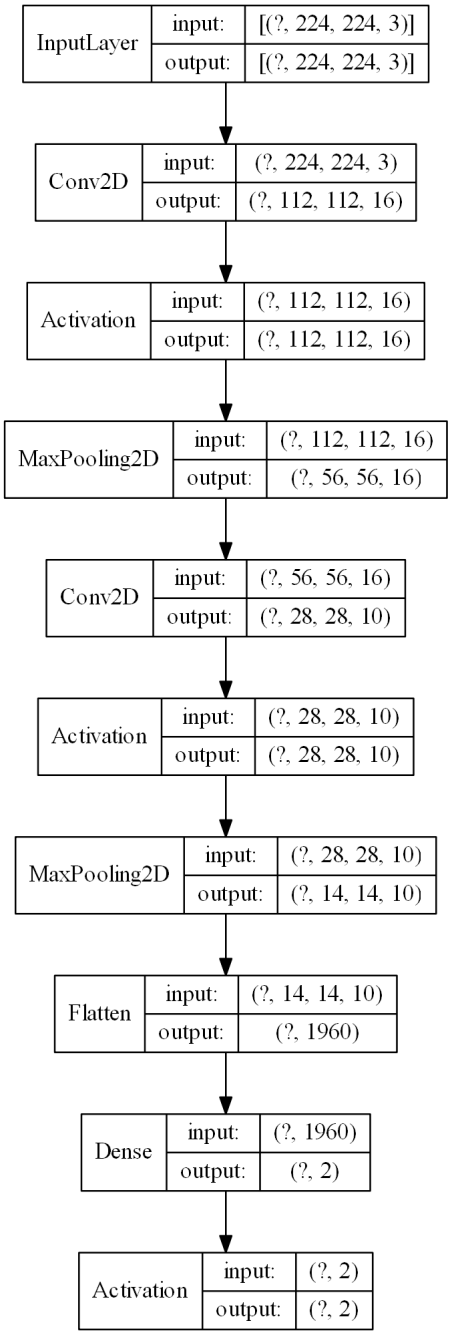
Seznam symbolů, veličin a zkratek

SGD	náhodné gradientní klesání – Stochastic Gradient Descent
RMSprop	propagace pomocí kvadratického průměru – Root Mean Square Propagation
RMS	kvadratický průměr – Root Mean Square
Adam	adaptivní odhad momentu – Adaptive Moment Estimation
Tanh	hyperbolická tangenta - Hyperbolic Tangent
ReLU	usměrněná lineární jednotka - Rectified Linear Unit
VOC	vizuální třídy objektů – Visual Object Classes
COCO	běžné objekty s kontextem – Common Objects In Context
YOLO	stačí jeden pohled – You Only Look Once
CNN	konvoluční neuronová síť – Convolutional Neural Network
RCNN	regionálně založená konvoluční neuronová síť – Region Based Convolutional Neural Networks
SUV	sportovní užitkové vozidlo – Sport Utility Vehicle
UAV	bezpilotní letoun – Unmanned Aerial Vehicle
RGB	červená, zelená, modrá – Red, Green, Blue
2D	dvojrozměrný – 2-dimensional
CIFAR-10	kanadský institut pro pokročilý výzkum – Canadian Institute For Advanced Research
SVM	strojová analýza s podporou vektorů – Support-vector Machines
FPS	snímek za vteřinu - Frame per Second
MNIST	modifikovaná databáze národního institutu pro standardy a technologie – Modified National Institute of Standards and Technology database
NIST	databáze národního institutu pro standardy a technologie – National Institute of Standards and Technology database

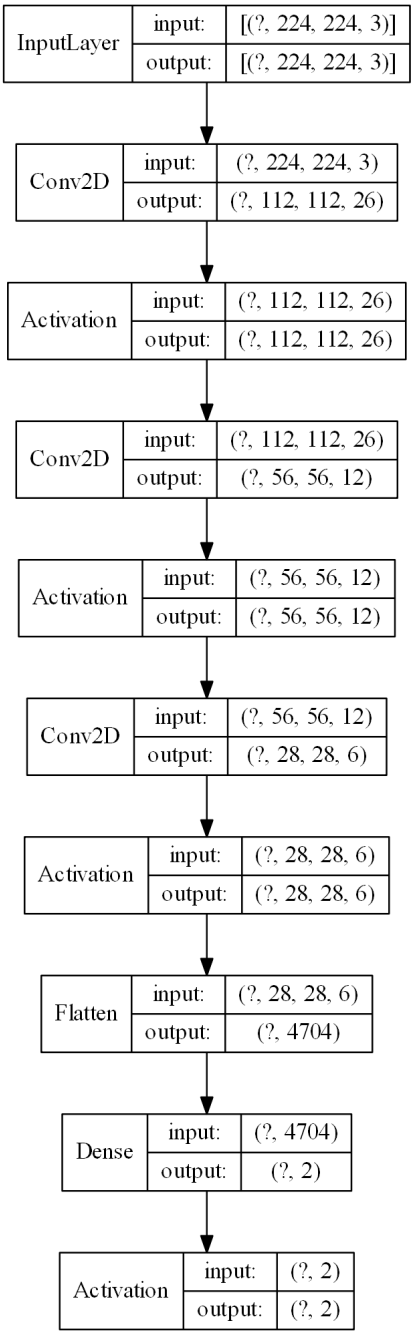
CUDA	jednotná architektura pro výpočetní zařízení -– Compute Unified Device Architecture
API	aplikační programovací rozhraní – Application Programming Interface
CPU	centrální procesorová jednotka — Central Processing Unit
GPU	grafická procesorová jednotka -– Graphic Processing Unit
TPU	tenzorová procesorová jednotka – Tensor Processing Unit
PC	osobní počítač – Personal Computer
USB	univerzální sériová sběrnice – Universal Serial Bus
IDE	vývojové prostředí – Integrated Development Environment
SPP	prostorové pyramidové podvzorkování – Spatial Pyramid Pooling
PAN	konvoluce založena na integrální křivce pro hloubkové grafové neuronové sítě – Path Integral Based Convolution for Deep Graph Neural Networks
∇	diferenciální operátor vektorové analýzy
∂	parciální derivace
α	konstanta pro Leaky ReLU
E_{test}	chyba testovací množiny
$E_{trénovací}$	chyba trénovací množiny
λ	komplexnost modelu
$G_c(i, j)$	2-D gabor filtr
$G_c(i, j)$	2-D gabor filtr
B, C	normalizující faktory ke zjištění
ms	milisekunda
GB	gigabyte
GHz	gigahertz
MHz	megahertz

MB	megabyte
mAP	střední průměrná přesnost – mean Average Precision

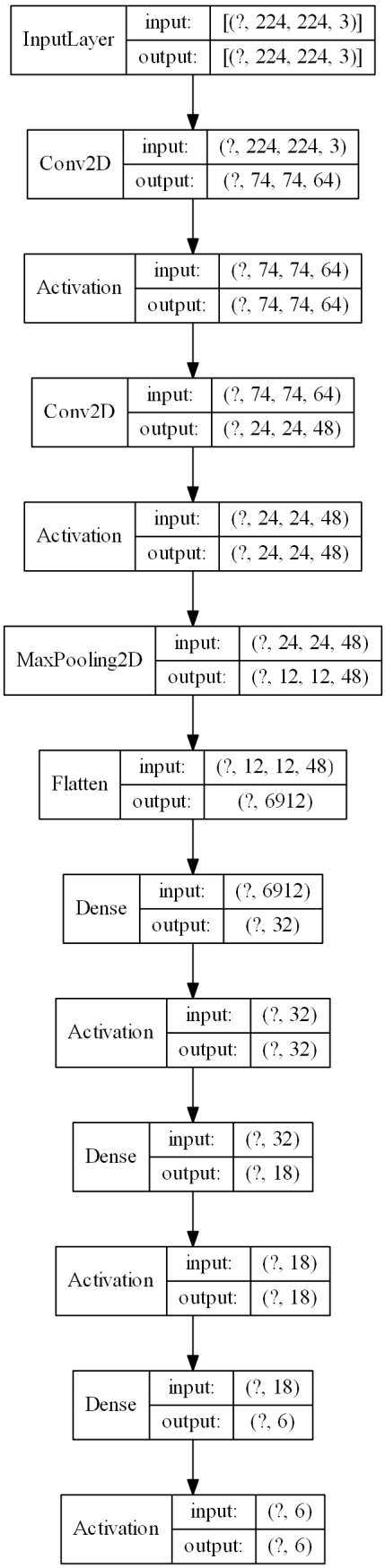
Seznam příloh



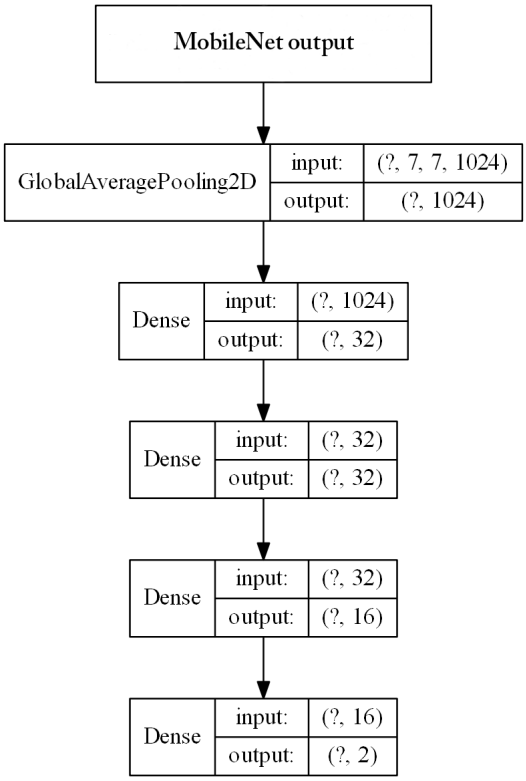
Obr. 17: Struktura vlastní konvoluční neuronové sítě.



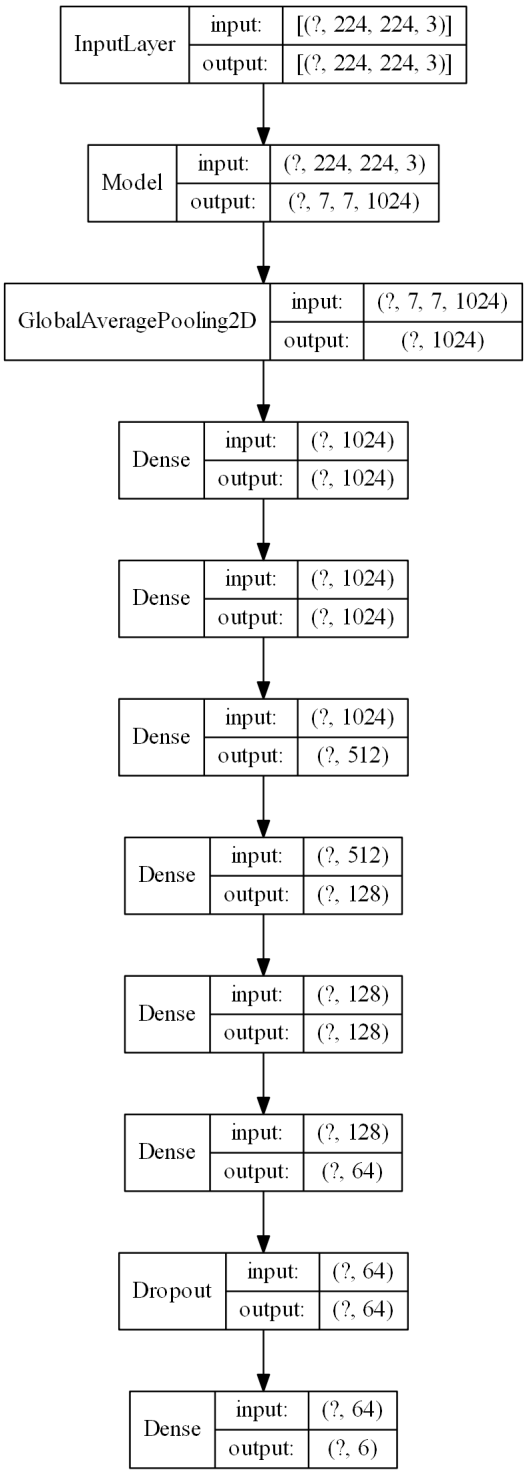
Obr. 18: Struktura konvoluční neuronové sítě s augmentací dat.



Obr. 19: Struktura konvoluční neuronové pro klasifikaci dopravních prostředků.



Obr. 20: Přidané vrstvy k modelu MobileNet.



Obr. 21: Struktura dotrénované neuronové sítě.