VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁNÍ DRUHU VOZIDLA V OBRAZE

RECOGNITION OF VEHICLE CLASS IN IMAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ROMAN ČABALA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAKUB ŠPAŇHEL
SUPERVISOR

BRNO 2020

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2019/2020

Zadání bakalářské práce

23009

Student: **Čabala Roman**
Program: Informační technologie
Název: **Rozpoznání druhu vozidla v obraze**
Recognition of Vehicle Class in Image
Kategorie: Zpracování obrazu
Zadání:

1. Seznamte se s metodami rozpoznávání/klasifikace objektů v obraze.
2. Pořiďte vhodnou datovou sadu pro klasifikaci druhu vozidla v obraze.
3. Vyhledejte metody hlubokého učení zaměřující se na problematiku klasifikace objektů, konkrétně klasifikace druhu vozidla.
4. Vyberte vhodné metody a experimentujte s nimi.
5. Vhodným způsobem vyhodnoťte vybrané metody a diskutujte dosažené výsledky.
6. Vytvořte plakát a video prezentující vaší práci, její cíle a výsledky.

Literatura:

- dle pokynů vedoucího

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Špaňhel Jakub, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 1. listopadu 2019

Abstrakt

Cieľom tejto bakalárskej práce je rozpoznať typ vozidla z obrazu pomocou neurónových sietí. Vozidlá sú rozdelené na šesť typov a to konkrétne - osobné auto, malá dodávka, dodávka, nákladné auto, kamión a autobus. Dátová sada bola vlastnoručne zozbieraná z videozáznamov, ktoré zaznamenávajú trajektóriu vozidiel. Následne bol zostrojený anotačný nástroj na anotovanie obrázkov. Na trénovanie sietí boli použité architektúry: VGG16, ResNet50, Xception, InceptionResNet-v2. Výsledkom práce je porovnanie architektúr. Všetky architektúry sa natrénovali a dosiahli výsledok nad 90%.

Abstract

The goal of this bachelor thesis is to recognize the type of vehicle from the image using neural networks. Vehicles are divided into 6 types, namely a car, a small van, a van, a mini truck, a truck and a bus. The data set was picked from videos that record the trajectory of the vehicles. Subsequently, an image annotation tool was built. The following architectures were used for network training: VGG16, ResNet50, Xception, InceptionResNet-v2. The result of the work is a comparison of architectures. All architectures were trained and achieved a result above 90%.

Kľúčové slová

klasifikácia druhu vozidla, Python, Tensorflow, Keras, VGG16, ResNet50, Xception, InceptionResNet

Keywords

vehicle type classification, Python, Tensorflow, Keras, VGG16, ResNet50, Xception, InceptionResNet

Citácia

ČABALA, Roman. *Rozpoznání druhu vozidla v obraze.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jakub Špaňhel

Rozpoznání druhu vozidla v obraze

Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Špaňhela. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Roman Čabala
27. mája 2020

Poďakovanie

Rád by som poďakoval svojmu vedúcemu práce Ing. Jakubovi Špaňhelovi za jeho odborné rady, čas a ochotu pri konzultáciách a za správne nasmerovanie pri riešení problému. Ďalej by som chcel poďakovať svojej rodine za podporu pri štúdii a pri písaní bakalárskej práce.

Obsah

Kapitola 1

Úvod

Automobilová doprava je najrozsiahlejšou dopravou na svete. Môžeme si všimnúť, že väčšina rodín vlastní aspoň jedno auto, iné rodiny aj viac. No automobilová doprava nie sú iba osobné autá. Patria tu kamióny, nákladné autá, dodávky a mnoho ďalších typov vozidiel. V súčasnosti si nevieme predstaviť život bez vozidiel, patria ku každodennému životu.

S týmto trendom prichádzajú aj problémy, napríklad znečisťovanie prostredia, hluk alebo postupné opotrebovanie ciest. Technológie môžu byt nápomocné. Napríklad na kamery sa môže nasadiť systém, ktorý bude monitorovať cestu a zaznamenávať typ vozidiel, ktoré cestou prejdú. Týmto krokom, by správa ciest poznala vyťaženosť jednotlivých ciest a konkrétne typy vozidiel, ktoré týmito cestami prejdú. Na základne toho, by mohla vydať vhodné opatrenia voči negatívnym vplyvom vozidiel pre danú cestu.

Technológia, ktorá je v súčasnosti veľmi populárna medzi vedcami aj verejnosťou, je umelá inteligencia. Ide o neurónové siete, ktoré sa učia postupne vyriešiť zložitý problém. Na náš problém sa dajú aplikovať klasifikačné modely neurónových sietí.

Preto sa v tejto bakalárskej práci budeme venovať klasifikácii vozidiel z obrazu. Vybral som 6 typov vozidiel a to konkrétne osobné auto, malá dodávka, dodávka, nákladné auto, kamión a autobus.

Ciele tejto práce sú:

- Popísať, čo je to neurónová sieť, ukázať a vysvetliť existujúce architektúry neurónových sietí, ktoré sa zaoberajú klasifikáciou objektov.
- Nazbierať vlastnú dátovú sadu a upraviť ju na natrénovanie neurónových sietí.
- Vybrať vhodné architektúry, natrénovať ich na rovnakých dátach a porovnať ich.

Práca je členená do šiestich kapitol. Kapitola 2 vysvetľuje základy neurónových sietí. V nasledujúcej kapitole 3 sú uvedené nástroje pre prácu s neurónovými sieťami. Tak isto v tejto kapitole je ukázaný prehľad architektúr neurónových sietí, ktoré sa zaoberajú klasifikáciou objektov. Kapitola 4 je venovaná zberu dát a jej úprave. V kapitole 5 je popísaný postup trénovania a výsledky.

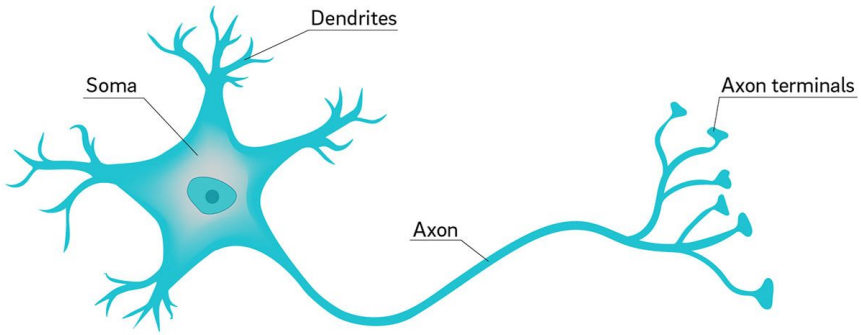
Kapitola 2

Umelá neurónová sieť

Umelá neurónová sieť je dátová štruktúra, ktorá je schopná reagovať na vopred neznáme vstupy [35]. V súčasnosti sa, čím ďalej tým viac, využívajú neurónové siete a to napríklad v oblastí počítačového videnia, rozpoznávania objektov, prekladu jazykov a mnoho ďalších. Nasledujúce podkapitoly obsahujú vysvetlenie pojmov: biologický neurón, model umelého neurónu, jeho podobnosť s biologickým neurónom a definíciu neurónových sietí [30].

2.1 Biologický neurón

Neurón je základný a funkčný prvok nervového tkaniva. Neurónové bunky sa špecializujú na spracovanie, prenos, uchovávanie a využívanie informácii. Informácie sa v nervovom systéme prenášajú vo forme zmien membránového potenciálu nervových buniek, neurónov. Biologický neurón sa skladá zo somy, čo je telo neurónu. Zo somy vychádzajú dva typy výbežkov a to dendrity a axon. Dendrity sú vstupmi, ktoré vedú vzruch smerom k bunke. Axon na druhej strane je výstup z bunky. Z axonu obvykle odbočuje rada vetiev tzv. terminálov, zakončených blanou, ktorá sa prevažne stretáva s výbežkami dentritov iných neurónov. Neuróny sú medzi sebou poprepájané väzbami, ktoré sa označujú ako synapsie. Synapsie slúžia k prenosu informácii medzi neurónmi. Pod pojmom neurón si môžeme teda predstaviť nejakú výpočtovú jednotku, ktorá na základe vstupov realizuje výstup [34].



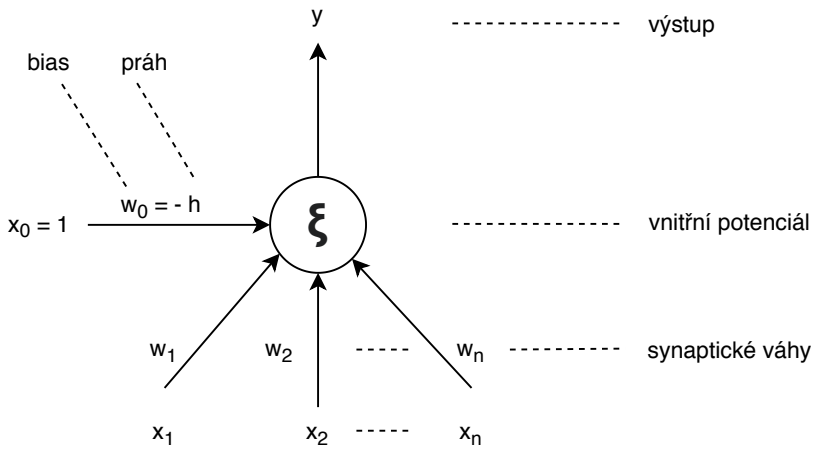
Obr. 2.1: Model biologického neurónu [13]

2.2 História umelého neurónu

Za tvorcov prvého umelého neurónu sa považujú vedci W. McCulloch a W. Pitts, ktorí v roku 1943 vymysleli abstraktný model neurónu. Číselné hodnoty parametrov v tomto modeli boli prevažne bipolárne (to znamená z množiny -1, 0, 1). Ukázali, že najjednoduchšie typy neurónových sietí môžu v princípe počítať ľubovoľnú aritmetickú alebo logickú funkciu. Tento model bol predchodcom ďalšieho modelu nazývaný perceptron z roku 1957. Perceptron zobecnil Frank Rosenblatt, ktorý tiež navrhol nový algoritmus na učenie. Hlavná myšlienka jeho učiaceho algoritmu je takáto: najskôr sa zaznamená odpoveď každého formálneho neurónu na daný podnet. Ak je odpoveď správna, nemodifikujú sa váhy. Ale ak je odpoveď daného neurónu nesprávna, potom sa modifikujú váhy všetkých aktivovaných vstupných synáps. Synapsie sa modifikujú nasledovne: ak má byť neurón aktívny a nie je, zväčšia sa. A naopak, ak má byť na výstupe neurónu 0 a nie je, zmenšia sa [30]. Táto idea modifikácie váh spojená na základe korekcie chýb tvorí základ mnohých algoritmov učenia s pomocou učiteľa, ktoré sa používajú dodnes. Krátko po objavení perceptronu Bernard Widrow so svojimi študentmi vyvinul ADALINE (Adaptive linear element). Tento model bol vybavený novým výkonným učiacim pravidlom, ktoré sa používa dodnes [44, 42].

2.3 Formálny neurón

Pod pojmom formálny neurón sa myslí matematický model biologického neurónu. Jeho schéma je na obrázku 2.2. Obecne má formálny neurón (ďalej iba neurón) n obecne reálnych vstupov $x_1, x_2, ..., x_n$, ktoré modelujú dendrity. Vstupy sú ohodnotené synaptickými váhami $w_1, w_2, ..., w_n$, ktoré označujú ich priepustnosť. Synaptické váhy môžu byť záporné, čo vyjadruje inhibičný charakter.



Obr. 2.2: Model formálneho neurónu [10]

Vnútorný potenciál neurónu predstavuje váženú sumu vstupných hodnôt:

$$\xi = \sum_{i=1}^{n} w_i \cdot x_i \tag{2.1}$$

Výstup z neurónu y sa počíta pomocou vnútorného potenciálu a aktivačnej funkcie o:

$$y = o(\xi) \tag{2.2}$$

Najjednoduchším typom aktivačnej funkcie je tzv. ostrá nelinearita, ktorá má tvar:

$$\sigma(\xi) = \begin{cases} 1 & \text{keď} \quad \xi \geq h \\ 0 & \text{keď} \quad \xi < h \end{cases} \tag{2.3}$$

kde h je hodnota prahu, tiež označovaná ako bias. Keď označíme $w_0 = -h$, $x_0 = 1$ potom bude ostrá nelinearita mať tvar:

$$\sigma(\xi) = \begin{cases} 1 & \text{keď} \quad \xi \geq 0 \\ 0 & \text{keď} \quad \xi < 0 \end{cases} \quad \text{kde} \quad \xi = \sum_{i=1}^{n} w_i \cdot x_i \tag{2.4}$$

Takto je definovaný perceptron, ktorý je popísaný v nasledujúcej kapitole 2.4.
Ako ďalšie aktivačné funkcie, ktoré sa používajú pri neurónových sieťach sú:

- Lineárna saturovaná funkcia

$$\sigma(\xi) = \begin{cases} 1 & \text{keď} \quad \xi > 1 \\ \xi & \text{keď} \quad 0 \leq \xi \leq 1 \\ 0 & \text{keď} \quad \xi < 0 \end{cases} \tag{2.5}$$

- Štandardná sigmoida

$$\sigma(\xi) = \frac{1}{1 + e^{-\xi}} \tag{2.6}$$

- Hyperbolický tangens

$$\sigma(\xi) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}} \tag{2.7}$$

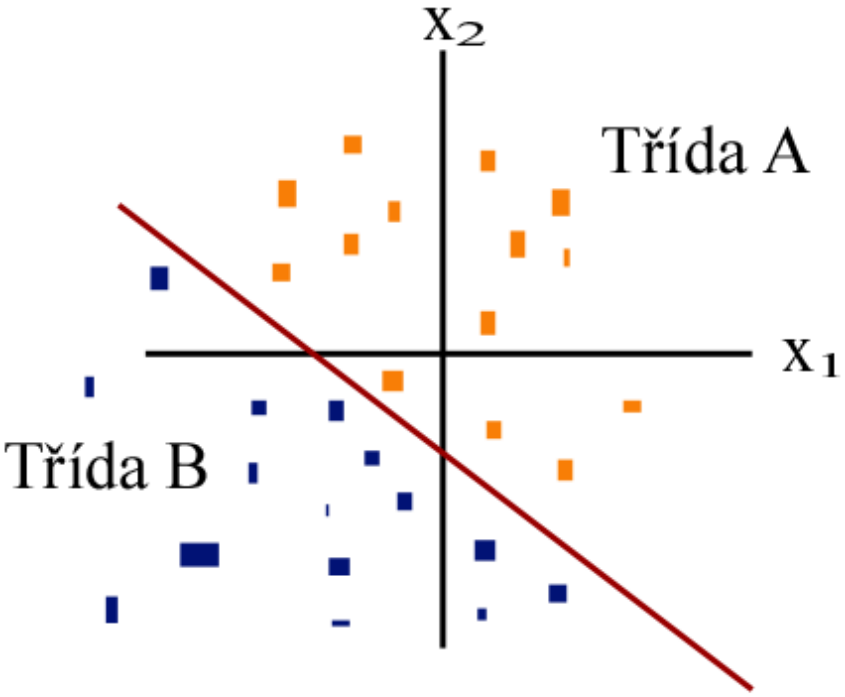
- ReLU

$$\sigma(\xi) = \begin{cases} 0 & \text{keď} \quad \xi < 0 \\ \xi & \text{keď} \quad \xi \geq 0 \end{cases} \tag{2.8}$$

2.4 Perceptrón

Základný princíp perceptrónu bol popísaný v kapitole 2.3. Jedná sa o formálny neurón, ktorý doplňuje vstupný vektor o nultú zložku, ktorá má pevnú hodnotu 1 a ktorej váha sa nastavuje na hodnotu záporného prahu [43]. Perceptrón je jeden z najdôležitejších modelov dodnes používaných [42]. Je určený na dichotomickú klasifikáciu, tj. rozdelenie do dvoch

tried, pri ktorých sa predpokladá, že triedy sú lineárne separovateľné v príkladovom priestore. Pod lineárnou separovateľnosťou dvoch tried rozumieme situáciu, keď existuje možnosť oddeliť objekty v príkladovom priestore pomocou nadroviny, napr: priamka v 2-rozmernom alebo rovina v 3-rozmernom priestore. Obrázok 2.3 nám to ukazuje.



Obr. 2.3: Lineárna separovanosť [15]

2.5 Neurónová sieť

Neurónovú sieť tvoria neuróny, ktoré sú medzi sebou poprepájané. Obecne môžeme neuróny poprepájať medzi ľubovoľným počtom neurónov, pričom okrem pôvodných vstupov môžu byť za vstupy brané aj výstupy iných neurónov. Počet neurónov a ich vzájomné poprepájanie v sieti určuje tzv. architektúru (topológiu) neurónovej siete. Neurónová sieť sa v čase vyvíja, preto je potrebné celkovú dynamiku neurónovej siete rozdeliť do troch dynamík a potom uvažovať tri režimy práce siete: organizačná (zmena topológie), aktívna (zmena stavu) a adaptívna (zmena konfigurácie). Jednotlivé dynamiky neurónovej siete sú obvykle zadané počiatočným stavom a matematickou rovnicou, resp. pravidlom, ktoré určuje vývoj príslušnej charakteristiky sieti v čase. V tejto podkapitole som vychádzal z knižky [44].

2.5.1 Organizačná dynamika

Organizačná dynamika prevažne predpokladá pevnú architektúru neurónovej siete, ktorá sa už nemení. Špecifikuje architektúru sieti a jej prípadnú zmenu. Zmena sa väčšinou uplatňuje v rámci adaptívneho režimu. Sieť je v prípade potreby rozšírená o ďalšie neuróny a príslušné prepojenia. Architektúru sieti rozlišujeme na dva typy: cyklická (resp. rekurentná) a acyklická (resp. dopredná) sieť. V prípade cyklickej sieti sa jedná o také usporiadanie neurónov,

ktoré sú zapojené v kruhu tzv. cyklus. Najjednoduchším príkladom cyklu je spätná väzba neurónu, kedy je výstup zároveň jeho vstupom.

Acyklická sieť je sieť, v ktorej sa nenachádzajú neuróny, ktoré sú zapojené v cykle. Neuróny v takejto sieti sa dajú rozdeliť do tzv. vrstiev, ktoré sú usporiadané tak, že spoje medzi neurónmi vedú len z najnižšej vrstvy do vyšších a obecne môžu preskočiť jednu alebo viacej vrstiev. Špeciálnym prípadom takejto vrstvy je tzv. viacvrstvová neurónová sieť. V takejto sieti rozdeľujeme vrstvy na vstupné, skryté a výstupné. Vstupná vrstva, inak označovaná ako nultá vrstva alebo dolná vrstva, sa skladá zo vstupných neurónov. Výstupná vrstva, inak označovaná ako posledná vrstva alebo horná vrstva, sa skladá z výstupných neurónov. Ostatné vrstvy sa nazývajú skryté a skladajú sa zo skrytých neurónov. Vrstvy sa číslujú od nuly, ktorá odpovedá vstupnej vrstve. Tá sa potom nepočíta do počtu vrstiev sieti.

2.5.2 Aktívna dynamika

Aktívna dynamika špecifikuje počiatočný stav siete a spôsob jej zmeny v čase pri pevnej topológii a konfigurácii. Na začiatku sa nastavia stavy vstupných neurónov a ostatné neuróny sú uvedené v počiatočnom stave. Po inicializácii stavu siete prebieha vlastný výpočet. Obecne sa uvažuje o spojitom vývoji stavu neurónovej siete, ktorá je obvykle v aktívnej dynamike zadaná diferenciálnou rovnicou. Väčšinou sa však predpokladá diskrétny čas. To znamená, že na začiatku sa sieť nachádza v čase 0 a stav siete sa mení iba v diskrétnom čase. V každom takom časovom kroku sa podľa daného pravidla aktívnej dynamiky vyberie jeden neurón (tzv. sekvenčný výpočet) alebo viacej neurónov (tzv. paralelný výpočet), ktoré aktualizujú svoj stav na základe svojich vstupov, to znamená stavov susedných neurónov, ktoré výstupy sú vstupmi aktualizovaných neurónov. Aktívna dynamika neurónovej siete taktiež určuje funkciu jedného neurónu, ktorej predpis je väčšinou pre všetky neuróny v sieti rovnaký. Aktivačná funkcia neurónu nemusí byť len funkcia, ktorá vychádza z biologického neurónu. Tak isto sa používajú aktivačné funkcie, ktoré vznikli buď matematickou invenciou alebo fyzikálnymi teóriami. Napríklad sa jedná o ostrú nelinearitu, saturovanú lineárnu funkciu, štandardnú sigmoidu alebo hyperbolický tangens. Všetky tieto aktivačné funkcie sú popísané v kapitole 2.3.

2.5.3 Adaptívna dynamika

Adaptívna dynamika špecifikuje počiatočnú konfiguráciu siete a akým spôsobom sa menia váhy v sieti v čase. Na začiatku sa nastavia váhy všetkých spojov siete na počiatočnú konfiguráciu (napr. náhodne). Po inicializácii prebieha vlastná adaptácia. Cieľom adaptácie je nájsť takú konfiguráciu siete vo váhovom priestore, ktorá by v aktívnom režime realizovala predpísanú funkciu. Adaptívny režim môže slúžiť k učeniu funkcie a to vtedy, ak aktívny režim siete sa využíva k vlastnému výpočtu funkcie siete pre daný vstup. Existuje veľa úspešných učiacich algoritmov pre rôzne modely neurónových sietí. Učiace algoritmy vieme rozdeliť do dvoch základných skupín, a to učenie s učiteľom a učenie bez učiteľa. Pri učení s učiteľom je obvykle zadaná tréningová množina dvojíc vstup/výstup. Model učiteľa pre vzorové vstupy siete informuje adaptívny mechanizmus o správnom výstupe siete. Niekedy učiteľ hodnotí kvalitu momentálneho skutočného výstupu siete pre daný vzorový vstup pomocou známky, ktorá je zadaná miesto požadovanej hodnoty výstupu siete. Jedná sa o klasifikované učenie. Na druhej strane učenie bez učiteľa obsahuje tréningovú množinu, ktorá je vstupom siete. Modeluje to situáciu, kedy nie je k dispozícii učiteľ. Niekedy sa

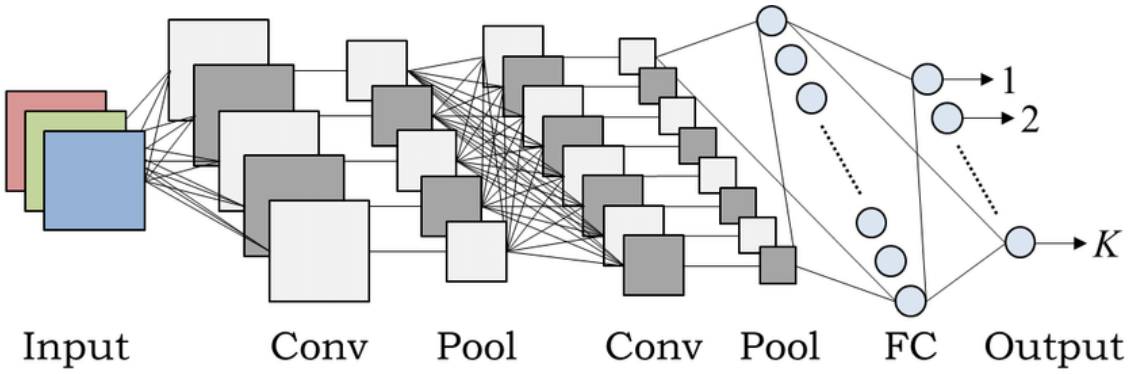
tejto adaptácii hovorí učenie bez učiteľa. Neurónová sieť v tejto adaptácii sama organizuje tréningové vzory a odhaľuje ich súborové vlastnosti.

2.6 Konvolučné neurónové siete

Táto podkapitola je venovaná konvolučným neurónovým sieťam, ktoré sa v súčasnosti využívajú pri rôznych úlohách rozpoznávania. Patrí sem napríklad rozpoznávanie číslic, objektov, spracovanie prirodzeného jazyka a podobne [33].

2.6.1 Základná charakteristika

Konvolučné neurónové siete alebo CNN vychádzajú z neocognitrovej siete, ktorú uviedol v roku 1987 K. Fukushimom. Neocognitron je hierarchická viacvrstvová neurónová sieť, ktorá bola navrhnutá pre rozpoznávanie ručne písaných znakov. Hlavnou výhodou neocognitronu je schopnosť identifikovať objekt na základe podobnosti vzorov bez ohľadu čiastočného posunu, rotácie alebo iného typu skreslenia [24]. CNN je špeciálnym prípadom viacvrstvových neurónových sietí. Taktiež, ako takmer všetky ostatné neurónové siete, využíva metódu spätného šírenia (anglicky back-propagation). Spätné šírenie je metóda na výpočet gradientu na základe chyby, pričom za chybu sa berie rozdiel medzi očakávaným a skutočným výstupom neurónovej siete [25]. Každá vrstva CNN môže slúžiť na inú úlohu. Medzi základné vrstvy patrí konvolučná vrstva, pooling vrstva a plne prepojená vrstva.



Obr. 2.4: Príklad architektúry konvolučnej siete [6]

Konvolučná vrstva je prvou vrstvou CNN. Táto vrstva v sebe obsahuje rôzne filtre. Každý filter má definovanú veľkosť a množinu váh. Napríklad typický filter na prvej vrstve môže mať veľkosť $5 \times 5 \times 3$ (čo značí 5 pixelov na dĺžku a výšku, 3 značí trojkanálovú hĺbku obrazu) [3]. To značí, že konvolučná vrstva pracuje s malou časťou obrázka, ktorá sa nazýva recepčné pole. Na tejto oblasti recepčného poľa aktuálny filter detekuje charakteristické črty, ktoré ovplyvňuje výstup z tejto vrstvy. Medzi charakteristické črty môžeme brať: hranu, roh alebo kruh. Výstupom konvolučnej vrstvy je tensor príznakových máp. Recepčné pole je postupne aplikované na celý obrázok. Zvyčajne existuje viacero recepčných polí, ktoré sú aplikované na vstupný obrázok. Jednotlivé recepčné polia sa líšia hodnotami váh vektorov. Váhy vektorov sa zdieľajú medzi neurónmi a tým sa urýchľuje učenie siete, lebo sa obmedzí počet výpočtov siete. Tak isto sa znižuje aj pamäťová náročnosť. Konvolučná vrstva nesie názov od konvolúcie, čo je operácia, s ktorou neuróny počítajú.

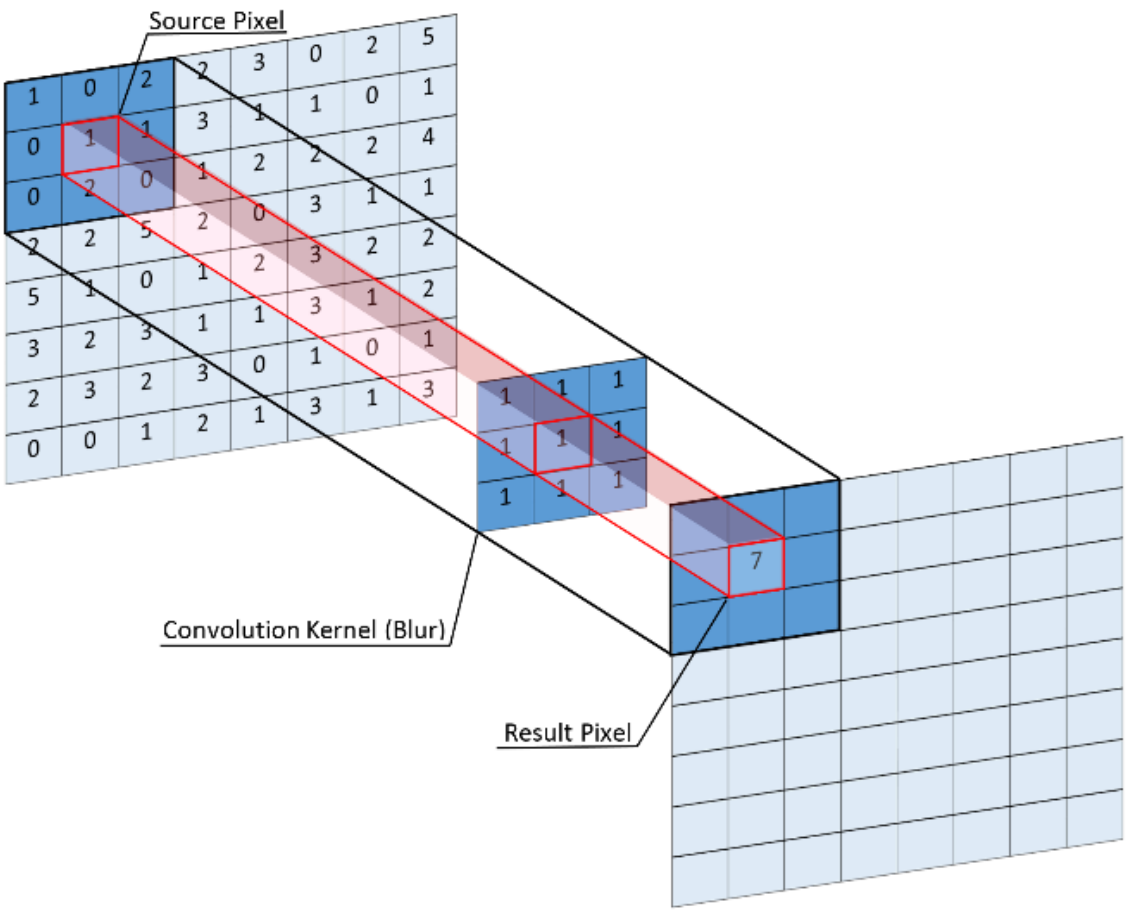
Matematicky zápis konvolúcie je:

$$I^{'} = f * h. \tag{2.9}$$

Vzorec pre výpočet jedného pixelu na súradniciach x a y pre diskrétnu 2D konvolúcie je:

$$(f * h)(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x - i, y - j) \cdot h(i, j), \tag{2.10}$$

kde symbol $*$ označuje konvolúciu a funkcia $h(x)$ reprezentuje konvolučné jadro, inak označované ako kernel. Konvolúcia funguje ako filter obrazu. Filtrom sa rozumie konvolučné jadro alebo kernel, ktorý má zvyčajne štvorcovú masku. Výstupom konvolúcie je pixel, ktorý je spočítaný ako súčet hodnôt konvolučného jadra s hodnotami vstupného obrazu. Operácia prebieha kým sa neprejde celý obraz. Priebeh konvolúcie je zobrazený na obrázku 2.5.



Obr. 2.5: Príklad konvolúcie jedného pixelu [4]

Nasledujúcou vrstvou CNN je pooling vrstva. V konvolučnej vrstve boli detekované charakteristické črty, ktoré v ďalšom vývoji siete sú menej dôležité. Nasledujúce vrstvy

potrebujú vedieť pozíciu relatívnu voči ostatným príznakom, aby dokázali rozpoznať vysokoúrovňové príznaky. Pre lepšiu detekciu príznakov je potrebné zmenšiť rozmery príznakovej mapy. Pooling vrstva sa stará o zmenšenie rozmerov príznakovej mapy. Ďalšia činnosť, ktorú vykonáva pooling vrstva je, že redukuje počet parametrov, ktoré treba trénovať. Tým sa zrýchli výpočtová rýchlosť a zníži sa pravdepodobnosť na preučenie siete. Najznámejším typom poolingu je tzv. max pooling, ktorý vyberá z recepčného poľa maximálnu hodnotu. Ďalším typom je tzv. average pooling, ktorý priemeruje hodnoty cez recepčné pole. V praxi funguje lepšie max pooling, average pooling sa využíva menej [3].

Ďalšie vrstvy, ktoré sa môžu nachádzať v CNN

Plne prepojená vrstva spája každý neurón v jednej vrstve s každým neurónom v inej vrstve. Slúži to predovšetkým na konci siete. Pri klasifikačných problémoch má plne prepojená vrstva počet kanálov rovný počtu kategórii, v ktorej je zadaná úloha. Je to z toho dôvodu, aby sieť vedela určiť pravdepodobnosť, do ktorej kategórie patrí obrázok [17].

Dropout vrstva slúži ako prevencia k pretrénovaniu siete. Nastaví sa určité percento neutrónov, ktoré sa počas trénovania môžu zahodiť. Tým zabudne časť vlastností, ktoré sa naučila a bude sa učiť na komplexnejších príznakoch. Síce sa počet iterácii približne zdvojnásobí, ale trénovací čas je kratší [17].

Batch normalizácia slúži na to, aby sa sieť stále učila. Odstraňuje extrémne hodnoty gradientu, ktoré môžu byť blízke nule alebo príliš vysoké. Takéto hodnoty spôsobuje zaseknutie siete, ktorá sa prestane učiť novým vlastnostiam. Táto vrstva sa v praxi používa medzi konvolučnou alebo plne prepojenou vrstvou a aktivačnou vrstvou. V nejakých prípadoch sa dropout vrstva nahradzuje batch normalizačnou vrstvou [27].

2.7 Zhrnutie

V úvode kapitola vysvetľuje biologický neurón, z ktorého vychádzajú neurónové siete 2.1. V nasledujúcej podkapitole je v krátkosti ukázaná história umelého neurónu 2.2. V podkapitolách 2.3 a 2.4 sú vysvetlené základy formálneho neurónu a perceptónu. Princíp perceptónu vychádza práve z formálneho neurónu.

Podkapitola 2.5 približuje podstatu neurónovej siete. Vysvetľuje základy stavby neurónovej siete a predovšetkým jej dynamiku. Dynamika je rozdelená do troch častí: organizačná, aktívna a adaptívna. Všetky tieto dynamiky sú v tejto podkapitole podrobnejšie popísané.

V ďalšej podkapitole 2.6 sú vysvetlené konvolučné neurónové siete, ktoré sa v súčastnosti využívajú pri rozpoznávaní objektov, či spracovaní jazyka. V úvode podkapitoly je vysvetlená základná charakteristika, následne je vysvetlený pojem konvolúcie. Tak isto sú vysvetlené jednotlivé vrstvy, ktoré sa v konvolučnej neurónovej sieti môžu nachádzať.

Kapitola 3

Nástroje a klasifikačné architektúry

V tejto kapitole budú predstavené nástroje na vytvorenie neurónových sietí a klasifikátorov, ktoré sú voľne dostupné verejnosti. Týchto nástrojov existuje veľké množstvo, preto budú spomenuté najpopulárnejšie medzi nimi. Ďalej sú v tejto kapitole spomenuté základné architektúry CNN, ktoré sa využívajú pre klasifikáciu objektov. Všetko sú to už vymyslené a predtrenované konvolučné siete, ktoré sa dajú natrénovať na vlastných dátach.

3.1 Nástroje na tvorbu neurónových sietí

TensorFlow

TensorFlow je softvérová knižnica, ktorá má otvorené zdrojové kódy (angl. *open source library*). Bola vytvorená v roku 2015 technologickým gigantom Google, kontrétne Google Brain tímom. Knižnica sa využíva pre numerické výpočty využívaných dátových vývojových diagramov (angl. *data-flow graphs*). Veľkou výhodou knižnice je, že sa dá používať na viacerých zariadeniach. Beží na skoro všetkých GPU, CPU, serveroch a mobilných telefónoch. Keď sa sieť natrénuje na PC alebo notebooku, tak natrénovaná sieť pôjde spustiť na inom PC, mobile alebo inom webovom rozhraní. Google používa na zlepšenie svojich služieb (gmail, fotky, vyhľadávací nástroj) práve neurónové siete, ktoré boli zostrojené na knižnici TensorFlow [18]. Tak isto túto knižnicu pre zlepšenie svojich služieb využívajú aj iné známe firmy ako Dropbox, Uber, AirBnB, DeepMind a pod. Aj vďaka tomu má TensorFlow veľkú komunitu ľudí a programátorov, ktorí stále vylepšujú a podporujú vývojárov knižnice [5]. Na druhej strane je veľmi nízko úrovňová knižnica a vyžaduje viac programovania oproti PyTorch alebo Keras knižnici. Podľa odborníkov jej ďalšou nevýhodou je jej pomalosť oproti iným nástrojom napr. CNTK alebo MXNet [37].

Keras

Keras je vysoko úrovňová API pre neurónové siete. Napísaná je v jazyku Python a pre svoj beh využíva ako backend knižnicu TensorFlow. Tak isto môže využívať ako backend knižnice CNTK alebo Theano. Hlavným dôvodom vzniku Keras-u je zjednodušiť tvorbu neurónových sietí. Minimalizuje počet úkonov pre bežné prípady a poskytuje jasnú a uskutočniteľnú spätnú väzbu pri chybe používateľa. Keras zachováva modularitu, to znamená, že existujúca architektúra sa dá ľahko vylepšiť, či zmeniť na riešenie iných problémov. Vďaka prívetivému

užívateľskému rozhraniu je dobrým nástrojom pre nováčikov v oblasti neurónových sietí [11]. Nevýhodou tejto siete pre niektorých užívateľov môže byť to, že neposkytuje toľko funkcií ako TensorFlow a užívateľ má menšiu kontrolu nad sieťou. Ďalšou nevýhodou je, že Keras je obmedzená na backend TensorFlow, CNTK alebo Theano knižníc [5].

Pytorch

PyTorch je nástupcom pythonovkej knižnice Torch, ktorá je veľkým konkurentom TensorFlow. Bol vyvinutý spoločnosťou Facebook v roku 2017. PyTorch ponúka dynamické výpočtové grafy, ktoré umožňujú spracovávať vstupy a výstupy s premenlivou dĺžkou [2]. To znamená, že umožňuje urobiť zmeny architektúry rovno pri procese. Existujú dve hlavné výhody PyTorchu - silná podpora zrýchlenia GPU na výpočet tenzorov a vytváranie hlbokých neurónových sietí na automatických systémoch založených na páske (angl. *tape-based autograd system*). Nevýhodami pre užívateľov môže byť zle napísaná dokumentácia a menšia podpora komunity oproti TensorFlow [36].

Caffe a Caffe2

Caffe je ďalší nástroj na tvorbu neurónových sietí. Je napísaný v jazyku C++ s rozhraním Pythona. Nástroj bol vyvinutý na Kalifornskej univerzite v Berkeley. Jeho hlavným využitím je spracovávanie obrázkov, nie spracovávanie textu, zvuku alebo časových údajov [2]. V roku 2017 firma Facebook vydala Caffe2, ktorý je predchodca prvej verzie. Caffe2 je postavený tak, aby pomohol vývojárom a výskumníkom trénovať veľké modely neurónových sietí a poskytovať AI na mobilných telefónoch [1]. Výhodou je, že vopred ponúka pretrénované modely na vytváranie demo aplikácií, je rýchly, ľahký a prispôsobiteľný. V máji roku 2018 sa Caffe2 spojila s PyTorch 1.0 do stabilnej verzie. Tým pádom PyTorch už obsahuje v sebe prvky Caffe2 [5].

MXNet

Ďalší voľno dostupný nástroj pre tvorbu neurónových sietí, je MXNet. Je vysoko škálovateľný, umožňuje rýchle natrénovanie modelov a podporuje viacero programovacích jazykov, vrátane: Python, C++, Javascript, Matlab, Perl. Veľkou výhodou nástroja je, že dokáže efektívne a paralelne pracovať s viacerými GPU a s viacerými strojmi. Veľké technologické spoločnosti, ako napríklad Microsoft, Intel a Amazon, využívajú práve MXNet na rozpoznávanie reči, rukopisu a na prognostiku [28]. Tento nástroj má niekoľko nevýhod. Nie je veľmi populárny vo výskumnej sfére a má menšiu komunitu oproti TensorFlow komunite [5].

CNTK

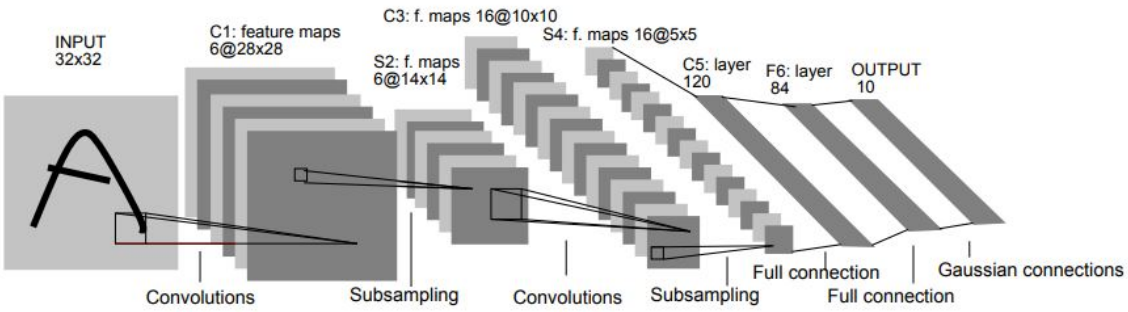
CNTK je skratka od *Microsoft Cognitive Toolkit* a je ďalší voľno dostupný nástroj, ktorý podporuje jazyky Python, C++, C# a Java. Je vyvinutý spoločnosťou Microsoft a je veľmi efektívny na natrénovanie hlasu, ručne písaného písma a na rozpoznávanie objektov. CNTK sa využíva vo viacerých známych programoch alebo technologických prístrojoch ako napríklad Skype, Xbox alebo Cortana. Nevýhodou je obmedzená podpora komunity [5].

3.2 Architektúry na klasifikáciu

Na svete existujú kompletné architektúry neurónových sietí, ktoré slúžia na riešenie určitých úloh. Táto práca sa venuje klasifikácii objektu z obrazu. Nasledujúce architektúry sa venujú práve klasifikácii objektov. Pre porovnanie architektúr medzi sebou sa využíva jedna databáza obrázkov, ktorá sa volá ImageNet. ImageNet obsahuje viac ako 20 000 kategórii objektov, kde v každej kategórii je priemerne 500 obrázkov. V celkovom počte obsahuje databáza vyše 15 miliónov obrázkov. Vďaka robustnosti databázy sa od roku 2010 každoročne koná medzinárodná súťaž ILSVRC, ktorá vyzýva vedcov, aby dosiahli najmenšiu chybovosť so svojou architektúrou pri detekcii a klasifikácii objektov [7]. Pre parameter chybovosti alebo presnosti sa využíva Top-1 presnosť a Top-5 presnosť. Top-1 nám udáva presnosť, kedy výsledok architektúry, teda výsledok s najvyššou pravdepodobnosťou, sa musí rovnať s očakávanou odpoveďou. Naopak Top-5 nám udáva, či očakávaná odpoveď je jednou z odpovedí v prvých piatich najvyšších pravdepodobnostiach architektúry [32].

3.2.1 LeNet-5

Najstaršou známou architektúrou, ktorá sa zaoberá rozpoznávaním je LeNet. LeNet bola zostrojená v roku 1998. Skladala sa dokopy zo 7 vrstiev. Táto sieť sa využívala na rozpoznanie písaných, strojových písmen a čísel. Vstupom do sieti boli obrázky o veľkosti 32×32 pixelov, ktoré boli dané do odtieňa sivej. Architektúra má 2 konvolučné a 3 plne prepojené vrstvy. LeNet-5 zostrojil tzv. štandardnú štruktúru, z ktorej vychádzali ďalšie architektúry [31]. Na obrázku 3.1 je zobrazený originálny model z roku 1998.



Obr. 3.1: Architektúra LeNet-5 [31]

3.2.2 AlexNet

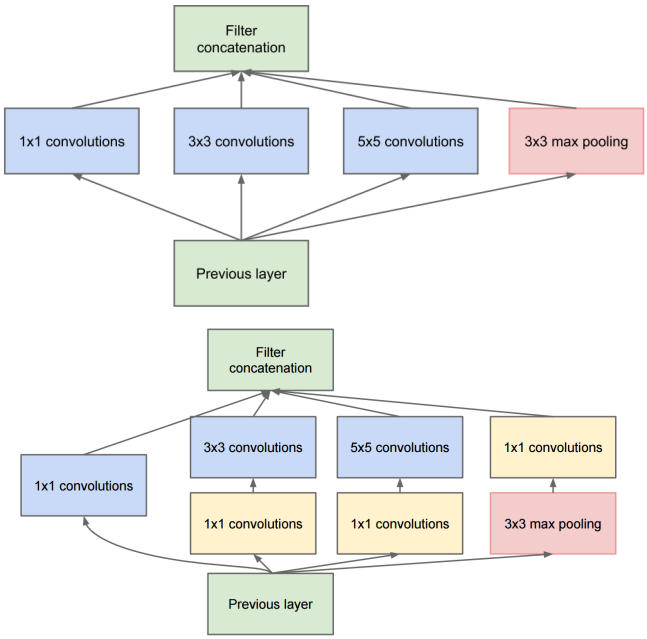
AlexNet architektúra bola vytvorená v roku 2012. Je veľmi podobná LeNet, ale sieť je hlbšia a obsahuje viac filtrov na jednotlivé vrstvy siete. Konkrétne sa skladá z 8 vrstiev, kde 5 vrstiev je konvolučných a 3 vrstvy sú plne prepojené. Ako aktivačná funkcia bola použitá ReLU. V tom čase bola štandardom tanh (*hyperbolický tangens*) ako aktivačná funkcia. Ďalšia výhoda tejto siete spočíva v trénovaní na viacerých GPU, tým sa dá trénovať väčší model a skracuje sa aj čas učenia. Architektúra AlexNet vyhrala v roku 2012 súťaž ILSVRC, lebo dokázala zredukovať top5-chybovosť na 15.3 %. To sa do tej doby nepodarilo žiadnej architektúre. V priemere mali všetky ostatné architektúry top-5 chybovosť okolo 26 % [29].

3.2.3 VGG

VGG ukazuje, že architektúra môže byť jednoduchá a úspešná, len musí byť hlboká. Vznikla v roku 2014 a rozdiel oproti AlexNetu bol v tom, že veľkosť filtrov v konvolučných vrstvách je o rozmere 3×3, pričom v AlexNete je 11×11. Síce VGG nevyhralo súťaž ILSVRC, ale poukázalo na to, že konvolučné neurónové siete sa majú zanechať jednoduché, ale zato je potrebné ísť viac do hĺbky [23]. Existujú verzie VGG16 a VGG19, pričom číslovka značí súčet konvolučných a plne prepojených vrstiev. Na obrázku 3.2 je zobrazený typ architektúry VGG16. Od architektúry VGG-16 všetky CNN siete išli hlbšie do hĺbky. Aktivačná funkcia bola rovnaká ako pri AlexNet a to ReLU [38].

Obr. 3.2: Architektúra VGG16 [20]

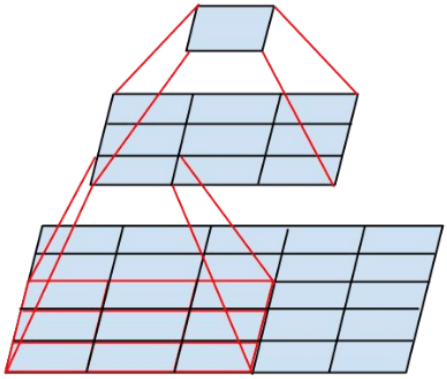
3.2.4 Inception-v1



Obr. 3.3: Porovnanie modelov Inception bez konv. 1×1 (hore) [9] a Inception s konv. 1×1 (dole) [8]

V rovnakom roku 2014 kedy vznikla VGG16 vznikla ďalšia architektúra a to Inception v prvej verzii. Jej druhý názov je GoogLeNet. Inception používa konvolúciu o veľkosti 1×1, ktorá slúži na zmenšenie rozmerov, na zníženie počtu operácii. Konvolúcia 1×1 sa pridáva pre zmenšenie vstupov pred 3×3 a 5×5 konvolúciu. Zatiaľ čo pri max pooling sa pridáva až za ňu. Na obrázku 3.3 je vidieť porovnanie modelu, ktorý využíva a nevyužíva konvolúciu 1×1. Tým sa odlišuje od architektúr ako AlexNet alebo VGG, v ktorých je veľkosť konvolúcie pevná pre každú vrstvu siete. V architektúre je dokopy 22 vrstiev. V rovnakom čase, kedy vznikla, to bol najhlbší model [40].

3.2.5 Inception-v2 a Inception-v3



Obr. 3.4: Mini-sieť nahrádzajúca 5×5 konvolúciu [14]

Architektúra Inception roku 2015 dostala nové verzie a to verzie Inception-v2 a Inception-v3. Obidve verzie boli predstavené v rovnakom vedeckom článku. Autori architektúry sa snažili zvýšiť presnosť siete a znížiť výpočtovú zložitosť.

Inception-v2

Hlavná myšlienka článku bola, že neurónová sieť má lepšie výsledky, keď sa rozmery vstupov nemenia drasticky. Autori tiež predpokladali, že využitie metód na faktorizáciu, môže pomôcť zefektívniť výpočty z hľadiska výpočtovej zložitosti. Zostrojili 3 modely, ktoré boli vylepšené z Inception-v1. V prvom modeli sa zmenila konvolúcia 5×5 na dve konvolúcie o veľkosti 3×3, pretože 5×5 konvolúcia je 2,78-krát viac náročná na výpočet ako 3×3 konvolúcia [41]. Na obrázku 3.4 je znázornené nahradenie 5×5 konvolúcie. Ďalší model využíva asymetrickú konvolúciu $n \times 1$. Napríklad použitie 3×1 konvolúcie nasledovanej 1×3 konvolúciou je ekvivalentné 3×3 konvolúcii. Táto zmena je o 33% výhodnejšia na výpočet ako jedna 3×3 konvolúcia [41]. Tretí model bol rozšírený o filtračné banky, ktoré urobili sieť širšiu ako hlbšiu. Keby model bol namiesto toho hlbší, došlo by k nadmernému zmenšeniu rozmerov a teda k strate informácii [41].

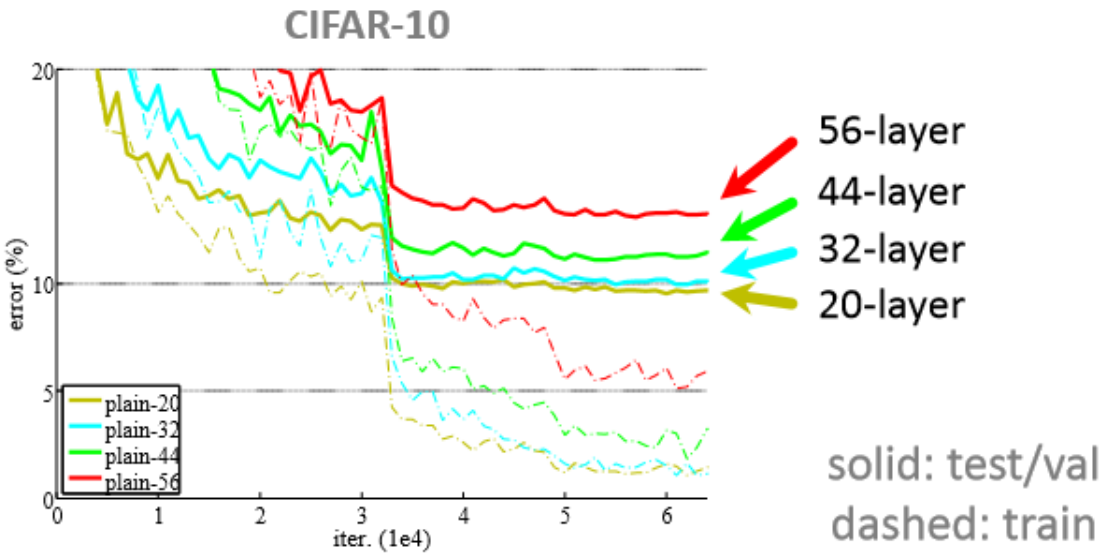
Inception-v3

Inception-v3 obsahoval všetko, čo jeho predchodza Inception-v2. Autori urobili veľa experimentov, aby vylepšili druhú verziu. Zamerali sa viac na zníženie drastických zmien rozme-

rov pri vstupe. Inception-v3 vylepšuje optimalizátor, stratové funkcie (z ang. *loss function*) a pridáva normalizáciu k pomocným vrstvám v pomocnej sieti (z ang. *auxiliary network*). Preto sa bežne druhá verzia nepoužíva [41].

3.2.6 ResNet

V tej dobe všetky architektúry išli veľmi do hĺbky. Experimentmi vedci zistili, že hlbšie siete dosahujú väčšie chyby ako menej hlboké siete [26], čo znázorňuje obrázok 3.5.



Obr. 3.5: Porovnanie chybovosti pre rôzne hlboké siete [19]

Preto v roku 2015 vyšla ďalšia architektúra na klasifikáciu a to ResNet. Táto architektúra prináša reziduálny blok, ktorého nová komponenta je tzv. skratka. Blok sa skladá z dvoch konvolučných vrstiev o veľkosti 3×3, aktivačnej funckie ReLU a zo skratky. Na obrázku 3.6 je znázornený reziduálny blok.

Skratka slúži na preskočenie jednej alebo viacerých vrstiev napr. z prvej vrstvy pomocou skratky sa môže skočiť na tretiu vrstvu. Skratky sa rozdeľujú na dva typy a to **identita** a **projekcia**.

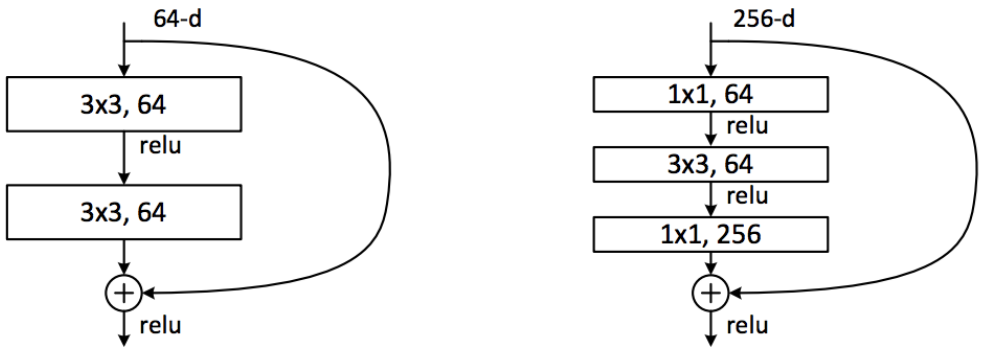
Identita nepridáva žiadny nový parameter. Priamo sa používa, keď majú vstup aj výstup rovnakú dimenziu. Vzorec 3.1 vyjadruje výpočet bloku, kde x a y je vstup a výstup. Funkcia $\mathcal{F}(\boldsymbol{x}, \{W_i\})$ predstavuje reziduálne mapovanie, ktoré sa má naučiť [26].

$$\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}, \{W_i\}) + \boldsymbol{x}. \tag{3.1}$$

Projekcia naopak slúži, ak nie je rovnaká dimenzia vstupu a výstupu. Pridáva extra nulový parameter na prispôsobenie dimenzie 1x1 konvolúciou pomocou vzorca 3.2, ktorý je podobný vzorcu pre identitu, len je pridaný lineárny parameter W_s pre prispôsobenie dimenzie [26].

$$\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}, \{W_i\}) + W_s\boldsymbol{x}. \tag{3.2}$$

Autori však vytvorili ďalší blok miesto reziduálneho bloku a to tzv. **bottleneck** blok. Rozdiel bol v počte vrstiev. Boli 3 vrstvy. Prvá mala veľkosť 1×1, druhá 3×3 a tretia 1×1. Vrstvy o veľkosti 1×1 slúžia najprv na zmenšenie a následne na zväčšenie dimenzie. Týmto sa zrýchli čas trénovania siete, pričom výsledky sú podobné aké dosahuje základný blok. V tomto bloku sa používa identita ako skratka. Obrázok 3.6 porovnáva reziduálny a bottleneck blok.



Obr. 3.6: Reziduálny blok (vľavo) a Bottleneck blok (vpravo) [16]

Vyšlo viac verzií ResNet, napr: ResNet50, ResNet101. Číslovka za názvom značí hĺbku - celkový počet vrstiev siete. ResNet152 vyhrala v roku 2015 ILSVRC súťaž, kde dosiahla chybovosť 3,57% na ImageNet datovej sade [26].

3.2.7 Xception

V roku 2016 bola predstavená nová architektúra a to Xception. Názov architektúry je spojením dvoch slov Extreme Inception [22]. A to z toho dôvodu, že architektúra vychádza z Inception a berie hypotézu do extrému. Pod extrémnou hypotézou sa rozumie najskôr použiť 1×1 konvolúciu na mapovanie krížovej korelácie (*z angl. map cross-channel correlations*). Potom sa samostatne zmapujú korelácie každého výstupného kanála. Táto extrémna forma je skoro identická hĺbkovo separovateľnými konvolúciami (*z angl. depthwise separable convolutions*). Preto architektúra zamieňa Inception modely práve hĺbkovo separovateľnými konvolúciami. Počet parametrov pri obidvoch sieťach je približne rovnaký. Autor svoju sieť natrénoval a porovnal ju z VGG-16, ResNet-152 a Inception V3. Všetky architektúry boli natrénované na sade ImageNet. Výsledky sú zobrazené v tabuľke 3.1.

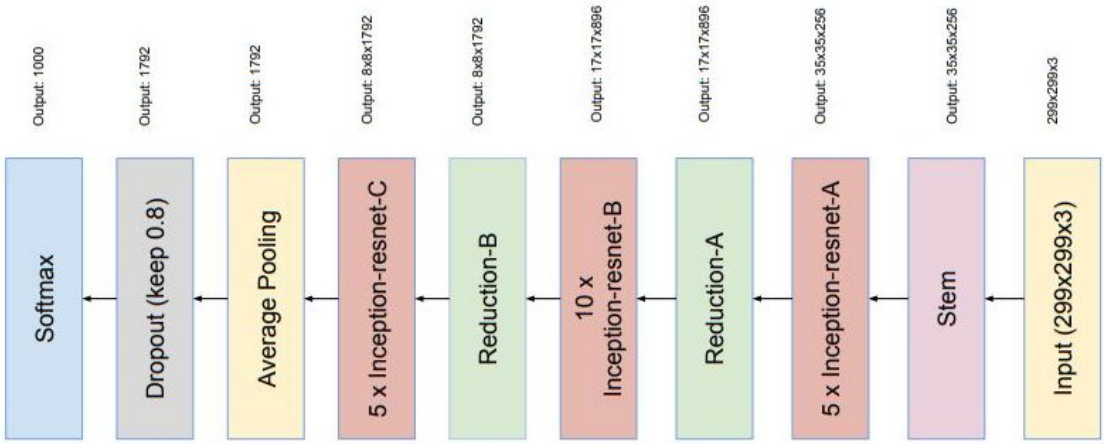
	Top-1 presnosť	**Top-5 presnosť**
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	**0.790**	**0.945**

Tabuľka 3.1: Tabuľka porovnávajúcu presnosť architektúr [22]

3.2.8 Inception-ResNet

V tom istom roku ako Xception vznikla aj nová architektúra, ktorá sa volá InceptionResNet. Bola predstavená v tomto vedeckom článku [39]. Autori predstavili rovno 3 nové architektúry a to konkrétne Inception-ResNet-v1, Inception-ResNet-v2 a Inception-v4. Všetky tieto architektúry vychádzajú z predchodcu a to z Inception-v3, dve z nich čerpajú aj z modelu ResNet. Inception-v4 sa zameriava na to, aby jednotlivé moduly neurónovej siete boli jednotnejšie. Autori tiež uviedli, že niektoré moduly v architektúrach sú viac zložitejšie ako potrebné. Riešením bolo, že koreň *(z ang. stem)* bol upravený tak, že označuje počiatočnú operáciu pred zavedením Inception blokov. Ďalej autori predstavili špeciálne reziduálne bloky, ktoré slúžia na zmenu šírky a výšky mriežky. Predchádzajúce verzie Inception nemali redukčné bloky, ale ich funkcionalita bola implementovaná.

Inception-ResNet siete sa inšpirovali výkonom architektúry ResNet. Rozdiel medzi dvomi verziami je v rozličných koreňoch. Obidve verzie majú rovnakú štruktúru pre jednotlivé modely a aj redukčné bloky sú rovnaké. Na obrázku 3.7 je znázornená schéma architektúry Inception-ResNet. Jediný rozdiel je v nastavení hyper-parametra. Inception-ResNet-v1 má výpočtovú cenu približne rovnakú ako architektúra Inception-v3. Naučí sa rýchlejšie, ale v závere má trochu horší výsledok ako Inception-v3. Inception-ResNet-v2 má výpočtovú cenu približne rovnakú ako Inception-v4, ale architektúra sa učí rýchlejšie a má o niečo málo lepšie výsledky ako Inception-v4. Autori ďalej v článku uviedli, že ak počet filtrov prekročí 1000, reziduálne spojenia začnú vykazovať nestabilitu a celá sieť umrie počas trénovania [39]. Vo vedeckom článku autori nepoužili batch normalizáciu, lebo chceli dosiahnuť trénovanie na jednej grafickej karte. Zistilo sa, že architektúry Inception-ResNet dosiahli lepšiu presnosť pri nižších epochách.



Obr. 3.7: Schéma architektúry Inception-ResNet [39]

3.3 Zhrnutie

Na začiatku táto kapitola ukázala súčasné nástroje pre prácu s neurónovými sieťami. Nástroje slúžia pre tvorbu neurónových sietí. Jedná sa o voľno dostupné knižnice alebo systémy. Podkapitola 3.1 vysvetlila základy, výhody a nevýhody medzi jednotlivými nástrojmi.

V ďalšej časti kapitoly 3.2 boli ukázané architektúry neurónových sietí, ktoré riešia problém klasifikácie objektov v obraze. Na začiatku podkapitoly sa vysvetlila metrika Top-1 a Top-5 presnosť, s ktorou sa porovnávali jednotlivé modely neurónových sietí. Ďalej v podkapitole je ukázaný vývoj architektúr v čase. Spomenuté sú tie architektúry, ktoré dosiahli veľmi dobré výsledky alebo prišli s niečím novým, čo zrýchlilo čas učenia siete, znížilo hardvérovú zátaž a pod. Architektúry sú v krátkosti popísané. Vysvetľuje sa ich základ, výhody a nevýhody oproti ostatným typom architektúr.

Kapitola 4

Dátová sada

Táto kapitola je zameraná na dáta, s ktorými sa pracuje vo zvyšku práce. Na úspešné natrénovanie neurónových sietí je potrebné mať veľkú dátovú sadu. Dátová sada pre klasifikáciu objektov v obraze by mala obsahovať čo najviac rozličných a unikátnych obrázkov. Táto práca sa zaoberá rozpoznávaním typov vozidla z obrazu. Konkrétne rozpoznávaním šiestich typov a to: osobné auto, mini dodávka, dodávka, nákladné auto, kamión a autobus. Dátová sada, s ktorou sa bude pracovať v práci, je súčtom mojej dátovej sady a sady kolegu Dalimila Rozpýrma, ktorý v rámci svojej práce vytváral rovnaký druh dátovej sady. Spoločne sme spojili dátové sady a vznikla jedna robustná sada.

4.1 Zber dát

Pre túto prácu dáta neboli stiahnuté z voľno dostupných zdrojov. Pre zber dát som používal kameru *Panasonic HC-V770* a statív. Kamera bola upevnená na statíve a nahrávali sa zábery, ktoré boli statické. Nahrával som premávku na troch rozličných miestach na Slovensku - v Košiciach. Všetky miesta, z ktorých som nahrával boli mosty. Jeden most bol nad rýchlostnou cestou. Vybral som tento most kvôli tomu, aby som do svojej dátovej sady nazbieral aj kamióny, nákladiaky a dodávky. Videa boli natáčané na fullhd rozlíšenie pri 30 snímkach za sekundu. V súčte som nahral 8 hodín video záznamu.

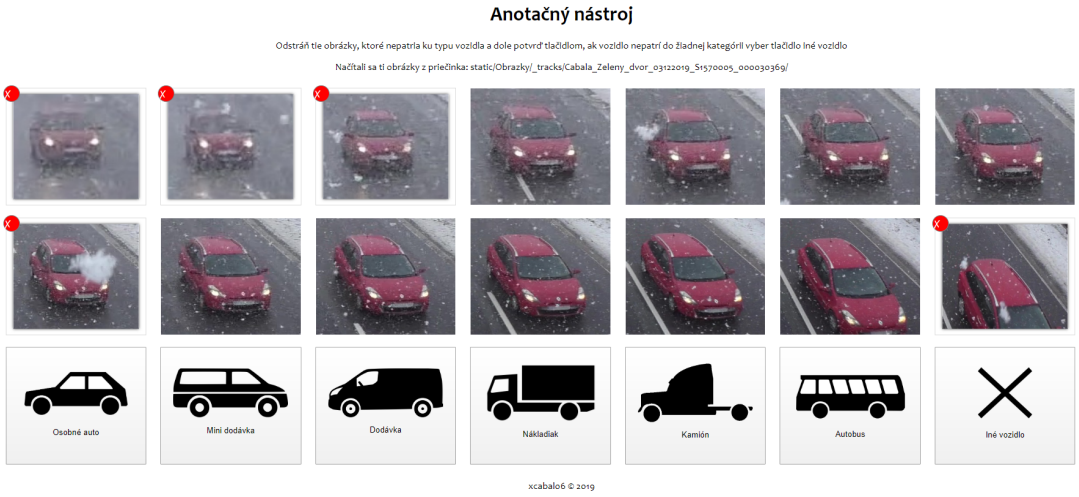
4.2 Spracovanie dát

Natočené videá som predal svojmu vedúcemu práce pánovi Ing. Jakubovi Špaňhelovi, ktorý nad všetkými videami spustil nástroj. Na detekciu vozidiel bol použitý detektor Mask R-CNN. Následne na vznik trajektórie vozidla sa využíva táto detekcia pomocou IOU trackera. Na obrázku 4.1 je ukázaný vstup a výstup nástroja. Na ľavej strane je ukázaný jeden snímok z videa a napravo je výstupný obrázok, ktorý nástroj vyhodnotil ako vozidlo na danom snímku. Takto postupne, zo snímok z videa, nástroj nájde vozidlo, oreže ho a vytvorí sadu obrázkov, ktoré zaznamenávajú trajektóriu daného vozidla. Dokopy zo všetkých videí vzniklo unikátnych 25 882 trajektórii vozidiel. To znamená, že ku každej trajektórii sa nachádza niekoľko fotografií vozidla. Dokopy dátová sada obsahovala 643 735 obrázkov. Niektoré z nich boli nepoužiteľné, preto všetky obrázky museli byť skontrolované a anotované. Nasledujúca podkapitola 4.3 vysvetľuje tento postup.

Obr. 4.1: Vľavo je jedna snímka z video záberu a vpravo výstupný obrázok z nástroja

4.3 Anotácia dát

V predchádzajúcej podkapitole 4.2 sa z video záznamu vytvorili obrázky, na ktorých sa neurónové siete dokážu učiť. Aby sa sieť dokázala učiť, treba každý obrázok anotovať, to znamená vytvoriť záznam ku každému obrázku a určiť typ vozidla, ktorý sa na danom obrázku nachádza. Pomocou anotácii sa sieť dokáže učiť, vie porovnať, či určila typ vozidla správne alebo nie a následne podľa toho automaticky zmení svoje parametre, ktoré určí za vhodné. Preto som zostrojil anotačný nástroj, ktorý je webovou aplikáciu pre možné a jednoduché anotovanie všade tam, kde je internetové pripojenie. Anotačný nástroj funguje tak, že si náhodne vyberie jeden priečinok s trajektóriou vozidla. Všetky priečinky trajektórii, teda 25 882 priečinkov, bolo napísaných v pomocnom textovom súbore *priecinky.txt*. Z tohoto súboru nástroj náhodne vybral jeden priečinok, skontroloval či sa v priečinku nachádza súbor *annot.json*. Ak sa nenachádzal, nástroj zobral všetky obrázky v danom priečinku a zobrazil ich užívateľovi vo webovom prehliadači. Keď sa súbor nachádzal v priečinku, vybral znova náhodne ďalší priečinok. Na obrázku 4.2 je zobrazený anotačný nástroj. Po načítaní obrázkov užívateľ uvidí trajektóriu jedného vozidla. Úlohou užívateľa je zo všetkých obrázkov vybrať tie, ktoré sú nesprávne. Pod nesprávnymi obrázkami sa rozumie: iný typ vozidla, s ktorým sa v práci nepracuje, nepodarený obrázok, ktorý nezaznamenáva vozidlo alebo obrázky, na ktorých je veľa rozličných vozidiel. Po vybraní nesprávnych obrázkov, užívateľ tento výber potvrdení jedným zo siedmych potvrdzovacích tlačidiel, ktoré sa nachádzajú naspodku aplikácie. Užívateľ vyberie to tlačidlo, ktoré odpovedá zvyšným neoznačeným obrázkom. Prvých šesť tlačidiel sú typy vozidla, s ktorými sa pracuje v tejto práci. Posledné tlačidlo nesie názov *Iné vozidlo*. Toto tlačidlo sa použije, ak celá trajektória vozidla obsahuje nesprávne obrázky, alebo obrázky iného typu vozidla, s ktorým sa nepracuje. V zápätí sa odošle požiadavka na server, ktorý v sebe obsahuje odkliknuté vozidlá, priečinok a typ vozidla. Server v danom priečinku vytvorí súbor *annot.json*, ktorý v sebe obsahuje cestu k priečinku, názvy dobrých obrázkov a k nim priradený typ vozidla. Následne automaticky vygeneruje užívateľovi novú trajektóriu.



Obr. 4.2: Ukážka anotačného nástroja

4.4 Zhrnutie

Táto kapitola bola venovaná dátam, ktoré sú veľmi dôležitou súčasťou trénovania neurónových sietí. V úvode kapitoly 4.1 je vysvetlený priebeh zberu dát, ktorý sa opieral o natáčanie vozidiel a zachytávanie ich trajektórie.

V nasledujúcej podkapitole 4.2 je popísaný postup spracovania dát. Z videí vzniklo 25 879 trajektórií vozidiel a dokopy 594 797 unikátnych obrázkov vozidiel.

Podkapitola 4.3 je zameraná na anotáciu dát. Kvôli obrovskému počtu obrázkov som zostrojil vlastný anotačný nástroj pre jednoduchú anotáciu obrázkov. Podkapitola vysvetľuje, čo je anotácia, detailne je popísaný priebeh anotácii obrázkov.

Kapitola 5

Implementácia a výsledky

V tejto kapitole je popísaný postup pri trénovaní neurónových sietí. V úvode je vysvetlené ako sa efektívne uložilo množstvo dát, aby neurónové siete mohli brať dáta z dátovej sady a učiť sa na nich. V kapitole 3 boli spomenuté základné nástroje a architektúry sietí pre klasifikáciu objektov. Táto kapitola bude obsahovať, ktoré z týchto nástrojov som využil a ktoré architektúry som si vybral pre natrénovanie na mojich dátach. Existuje veľmi veľa architektúr, tie najdôležitejšie som už spomenul. Z nich som si vybral 4, ktoré natrénujem na rovnakých dátach a porovnám ich. Vybraté siete sú: VGG16, ResNet-50, Inception-ResNet-v2, Xception. VGG16 a ResNet50 som vybral na základe toho, že ide o jedné z prvých architektúr, ktoré sa zaoberajú klasifikáciou objektov v obraze. Architekúry Inceptino-ResNet-v2 a Xception som vybral na základe úspešnosti predikcie, ktoré dosiahli na imagenet datasete. Všetky siete boli natrénované na osobnom počítači na grafickej karte Nvidia GeForce GTX 1070, ktorá obsahuje 8 GB GDDR5 video pamäte.

5.1 Nástroje a nastavenia dát

Pre moje zadanie som si vybral programovací jazyk Python, kvôli veľmi dobrej podpore a obsahujúcim nástrojom, pre tvorbu a prácu s neurónovými sieťami. Ako hlavný nástroj som si vybral Keras API, ktorá využíva backend knižnicu TensorFlow. V prvom kroku som vytvoril skript *json_parser.py*, ktorý postupne načítava všetky súbory *annot.json* z priečinkov trajektórii vozidiel. Po načítaní súbora skript upraví údaje do formátu:

```
{"images_path": path,
  "vehicle_type": type,
  "instances": {"path": [img_1, img_2, ...]}
}
```

Všetky upravené anotácie sa uložili v jednom súbore. Skript v nasledujúcom kroku odstráni všetky údaje, v ktorom sa nachádza *invalid* trajektória. To znamená, odstráni záznamy obrázkov, ktoré boli počas anotovania označené za chybné. V poslednom kroku skript načíta tento súbor a pomocou knižnice *Pandas* sa vytvorí tabuľka *DataFrame*. Túto tabuľku následne uloží. Takto uložená tabuľka slúži k ľahkému spracovaniu obrázkov.
Nasledujúci skript *dataframe_parser.py* načíta uloženú tabuľku. Pristupuje k nej po riadku a ukladá si hodnoty do 3 rozličných polí - na indexy, obrázky a na typy vozidiel. Pole na indexy postupne indexuje každý obrázok. Pole na obrázky v sebe zaznamenáva konkrétny obrázok s plnou cestou, na ktorom sa bude neurónová sieť učiť. Pole na typy vozidiel v sebe obsahuje priradený typ vozidla ku konkrétnemu obrázku. Všetky tri polia sú rovnako in-

dexované. To znamená, pri rovnakom indexe v každom poli sa dostaneme ku správnemu priradeniu obrázka k typu. Pole indexov slúži na to, aby neurónová sieť vedela jednoduchšie pracovať s dátami.

5.2 Dáta generátor

Dátovú sadu som upravil na trénovaciu sadu a testovaciu sadu. Trénovacia sada obsahuje 80% všetkých obrázkov a testovacia 20% všetkých obrázkov. Ďalej som to rozdelil práve tak, aby z každej kategórie vozidiel bolo práve 20% v testovacej sade. Tabuľka 5.1 ukazuje početnosť dátových sád. Spravil som to z dôvodu, že bol veľký nepomer medzi typom osobného auta a zvyšnými typmi vozidiel. Príčinou môže byť to, že osobné autá sú rozšírenejšie vo svete, a preto som ich zaznamenal viackrát, aj keď som natáčal nad rýchlostnou cestou. Na začatie trénovania neurónových sietí je dôležitý generátor obrázkov. Generátor obrázkov slúži na dodávanie obrázkov neurónovej sieti v reálnom čase. Pri tvorbe takého generátora som vychádzal z [21]. V *Data_generator.py* je implementovaná trieda *DataGenerator*. Trieda obsahuje niekoľko metód. Na začiatku metóda *__init__* inicializuje pole indexov, obrázkov a typov vozidiel. Ďalej inicializuje batch size, počet klasifikácií (v tomto prípade 6), dimenziu obrázka a počet kanálov obrázka. Všetky obrázky sú farebné, preto sa počet kanálov určil na 3 podľa RGB. Poslednou časťou je parameter *shuffle*, ktorý je boolean hodnotou. Trieda *__len__* vracia hodnotu počet $\frac{poet\_obrazkov}{batch\_size}$. Hodnota slúži ďalej na vytvorenie batchov, na ktorých sa sieť bude učiť. Trieda *__data_generation* generuje dáta. Zoberie konkrétny obrázok s konkrétnym typom. Obrázok upraví do formátu ako si to vyžadujú konkrétne modely neurónových sietí. Následne metóda vracia upravený obrázok a jeho typ. Nasledujúca metóda *__map_labels* obsahuje slovník a mapuje typ vozidla na číslo, aby sieť vedela určiť typ. Metóda *__getitem__* si generuje indexy batchov a hľadá ich v indexoch obrázkov. Následne pre obrázok a pre typ volá metódu *__data_generation*. Výstupom je obrázok s typom. Posledná metóda *on_epoch_end* slúži na načítanie indexov obrázkov. Ak sa pri inicializácii nastavil parameter *shuffle* na true, tak sa indexy zamiešajú.

Typ vozidla	**Celkový počet obrázkov**	**Počet obrázkov trénovacej sady**	**Počet obrázkov testovacej sady**
Osobné auto	423 280	338 626	84 654
Malá dodávka	24 785	19 828	4 957
Dodávka	41 135	32 909	8 226
Nákladiak	17 083	13 667	3 416
Kamión	27 400	21 921	5 479
Autobus	61 114	48 892	12 222
Súčet	**594 797**	**475 843**	**118 954**

Tabuľka 5.1: Tabuľka početnosti dátovej sady

5.3 Prvý experiment

VGG16

V prvom trénovaní som zobral celú dátovú sadu, ako je popísaná v tabuľke 5.1. Na trénovanie som použil optimizer *Adam*, s predvolenou *learning rate*, ktorá je rovná 0,001.

Loss funkciu som vybral *categorical_crossentropy*, pretože ma zaujíma pravdepodobnosť správneho výberu z viacerých kategórií. Vybrané architektúry som prevzal zo stránky [12]. Na všetkých sieťach bolo potrebné zmeniť poslednú vrstvu siete, aby sa učili a predikovali výsledky na počte typov vozidiel (v tomto prípade 6). Tento krok je dôležitý z dôvodu, že všetky siete boli natrénované na datasete imagenet, ktorý obsahuje 1000 objektov na rozpoznávanie. Poslednú vrstvu som pridal typu *dense*. Obsahovala parametre počtu predikovaných typov vozidiel, aktivačnú funkciu *softmax* a bola plne prepojenou vrstvou. Všetky siete mali počiatočnú váhu trénovania imagenet. Prvé dve siete, ktoré som trénoval boli VGG16 a ResNet50. Tieto siete berú na vstup obrázok o veľkosti 224×224 pixelov. Pre tieto siete som vybral batch veľkosť 32. Skúšal som vyššiu batch veľkosť, ale grafická karta nepustila učenie, lebo si nevedela alokovať potrebné miesto. Pre tieto siete sa veľkosť jednej epochy skladala z 14 870 krokov. Prvú som trénoval architektúru VGG16. Veľkosť VGG16 je 528 MB. Je to najväčšia sieť zo spomínaných sietí. Sieť som nechal trénovať 10 epoch. Jedna epocha sa trénovala približne 6500 sekúnd. Celkové trénovanie zabralo približne 18 hodín. Trénovanie som vypol z dôvodu, že presnosť predikcie sa nezvyšovala a loss funkcia klesala veľmi pomaly. Výsledkom, po 10. epoche, bola presnosť trénovania 0,7166 a loss funkcia mala hodnotu 1,0137, pričom presnosť sa nemenila od 4. epochy.

ResNet50

Z nepriaznivého výsledku som prešiel na architektúru ResNet50. Parametre som nechal rovnaké ako pri sieti VGG16. ResNet50 má veľkosť 98 MB. Je vidieť veľký rozdiel vo veľkosti oproti architektúre VGG16. Táto sieť sa učila o niečo rýchlejšie. Jedna epocha trvala približne 5200 sekúnd. Trénovanie prebiehalo lepšie. Loss funkcia klesala každou epochou a presnosť sa zvyšovala. Túto sieť som nechal trénovať 20 epoch, čo zabralo približne 29 hodín. Výsledkom trénovania, po 20. epoche, bola presnosť 0,9941 a loss funkcia mala hodnotu 0,0176. Presnosť učenia bola veľmi optimistická.

Xception

Ďalšie dve siete, ktoré som trénoval, boli siete Xception a Inception-ResNet-v2. Tieto siete sa odlišujú od predchádzajúcich tým, že vstupný obrázok do siete má veľkosť 299×299. Batch veľkosť som musel zmeniť z 32 na 16. Dôvodom bolo to, že vstupné obrázky sú väčšie a grafická karta už nedokázala alokovať toľko miesta na trénovanie. Preto sa trénovanie ani nespustilo. Po zmene batch veľkosti sa trénovanie spustilo. Zmenou batch veľkosti sa zmenila aj veľkosť počtu kroku jednej epochy. Jedna epocha sa teraz skladala z 29 740 krokov. Tento počet mal za následok dĺžku trénovania. Veľkosť expection je 88 MB. Jedna epocha na sieti Xception sa trénovala približne 17 500 sekúnd. Loss funckiu a optimizer som ponechal rovnakú ako pri predchádzajúcom trénovaní VGG16 a ResNet50. Sieť som nechal trénovať 10 epoch. Trvalo to približne 50 hodín na mojom osobnom počítači. Po každej epoche loss funkcia klesala a presnosť rástla. V 10. epoche sieť mala presnosť učenia 0,9927 a loss funkcia mala hodnotu 0,0222.

Inception-ResNet-v2

Pri trénovaní Inception-ResNet-v2 som nechal rovnaké parametre ako pri Xception. Veľkosť architektúry je 215 MB. Sieť sa učila o niečo rýchlejšie ako spomínaný Xception. Jedna epocha trvala približne 16 500 sekúnd. Sieť som nechal trénovať tiež 10 epoch a to z dôvodu, že presnosť po 10. epoche bola 0,9925. Loss funkcia mala hodnotu 0,0243.

Výsledky

Po spomínanom natrénovaní sietí som všetky siete nechal vyhodnotiť nad testovacou sadou. V tabuľke 5.1 je znázornený počet obrázkov testovacej sady. Všetky siete, okrem Xception, mali rovnakú úspešnosť a to **71,16**%. Xception mala presnosť **10,27**%. Tento výsledok bol spôsobený tým, že všetky siete, okrem Xception, určili každý obrázok z testovacej sady ako osobné auto. Xception naopak všetky obrázky z testovacej sady určil ako autobus. Osobných áut sa v testovacej sade nachádzalo 84 645 z 118 944 celkového počtu vozidiel a autobusov sa nachádzalo 12 222. Všetky siete sa pretrénovali a určovali iba jeden typ vozidla. Tento neúspech trénovania mohol byť spôsobený nevyváženou dátovou sadou. Predovšetkým pomer osobných áut oproti ostatným typom vozidiel bol veľmi vysoký. Ide však o model reálneho sveta, kedy je na cestách pomer osobných áut oproti ostatným vozidlám vysoký. Druhým možným dôvodom zlého natrénovania sietí, bol nesprávne zvolený optimizer s learning rate. Tabuľka 5.2 zobrazuje úspešnosť jednotlivých architektúr po prvom experimente.

	Presnosť
VGG16	71,16%
ResNet50	71,16%
Xception	10,27%
Inception-ResNet-v2	71,16%

Tabuľka 5.2: Tabuľka úspešnosti architektúr po prvom experimente

5.4 Druhý experiment

Vzhľadom na neúspech prvého experimentu, som v druhom experimente zmenil veľkosť dátovej sady. Zmenil som iba počet osobných áut. Stále som však zachoval väčší pomer osobných áut, aby som sa priblížil reálnemu svetu. Počet osobných áut som zredukoval z 423 280 na 100 000. Následne som upravil aj počet obrázkov testovacej sady. Početnosť zvyšných typov vozidiel som zmenil a z testovacej sady som pridal viac obrázkov do trénovacej sady, kvôli malému počtu obrázkov. Početnosť novej dátovej sady je zobrazená v tabuľke 5.3.

Typ vozidla	**Celkový počet obrázkov**	**Počet obrázkov trénovacej sady**	**Počet obrázkov testovacej sady**
Osobné auto	100 000	73 920	26 080
Malá dodávka	24 785	20 949	3 836
Dodávka	41 135	34 733	6 402
Nákladiak	17 083	14 166	2 917
Kamión	27 400	22 723	4 677
Autobus	61 114	50 731	10 383
Súčet	**271 517**	**217 222**	**54 295**

Tabuľka 5.3: Tabuľka zredukovanej početnosti dátovej sady

Trénovanie

Ďalšiu zmenu, ktorú som urobil, bola zmena optimizera. Pôvodný optimizer *Adam* som zmenil na *SGD*. *Learning rate* hodnotu som nechal pôvodnú 0,01. Loss funkciu som zachoval *categorical_crossentropy*. Na začiatku som trénoval architektúru ResNet50. Zmenšením početnosti dátovej sady, sa zmenšil aj čas učenia siete. Pri architektúrach VGG16 a ResNet50 bol počet krokov jednej epochy 6788, pričom batch veľkosť bola 32. Zmenil som spôsob vyhodnocovania výsledkov a natrénoval som iba niekoľko prvých epoch. Následne som načítal uložené váhy a spustil testovanie. ResNet50 som trénoval 5 epoch. Trénovanie jednej epochy, po zmene veľkosti dátovej sady, trvalo približne 2300 sekúnd. Išlo o značné zrýchlenie oproti prvej robustnej dátovej sade. Celkovo 5 epoch sa trénovalo približne 3,5 hodiny. Presnosť trénovania bola 0,9542 a loss 0,1851. Po natrénovaní som sieť otestoval na testovacej sade. Výsledok bol **14,69**%. V pomocnom výpise z testovania som videl, že sieť určila 37 972 obrázkov ako dodávky, 15 728 obrázkov ako autobus a 572 obrázkov ako osobné auto. Z tohto výpisu bolo vidieť, že sieť sa nedokáže správne učiť. Ďalšou architektúru, ktorú som skúsil natrénovať pri týchto podmienkach, bola sieť Xception. Sieť som taktiež trénoval 5 epoch. Batch veľkosť musela byť nastavená na 16, lebo pri 32 sa trénovanie nespustilo. Preto sa počet krokov jedne epochy zväčšil na 13 576. Trénovanie jednej epochy trvalo približne 7200 sekúnd. Po natrénovaní som sieť otestoval. Výsledok bol **23,49**%. Táto architektúra určila vždy iba autobus a dodávku.

Zhrnutie

Týmito pokusmi som zistil, že siete sa nedokážu správne učiť. Zvyšné dve siete som ani netrénoval, na základe neúspechu z trénovania ResNet50 a Xception. Oproti prvému experimentu nastalo zlepšenie, lebo siete už neurčovali iba jeden typ vozidla. Zmena veľkosti dátovej sady a zmena optimizera pomohla, ale nie dostatočne.

	Presnosť
ResNet50	14,69%
Xception	23,49%

Tabuľka 5.4: Tabuľka úspešnosti architektúr po druhom experimente

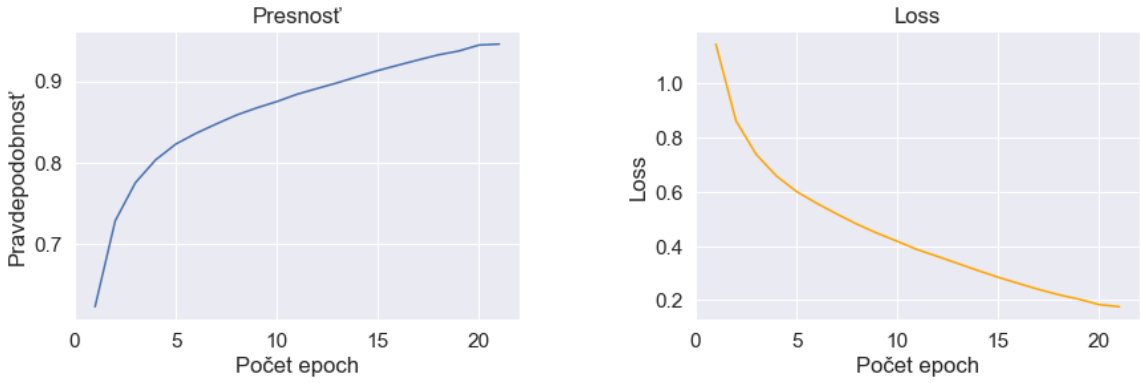
5.5 Tretí experiment

V tomto experimente som dátovú sadu ponechal, ako v druhom experimente 5.3. Teraz som sa sústredil na vhodné zvolenie optimizera s *learning rate*. Rozhodol som sa ponechať optimezer SGD. *Learning rate* som však upravil na hodnotu 0,0001. Siete som nechal trénovať pár epoch, aby som zistil či sa trénujú.

5.5.1 VGG16

Prvú sieť, ktorú som trénoval bola VGG16. Batch veľkosť som nastavil na 32. Po piatich epochách som sieť otestoval na testovacej sade a výsledkom bola úspešnosť 83,05%. Týmto výsledkom som zistil, že sieť sa začala učiť a optimizer s danou *learning rate* funguje. Sieť som pravidelne testoval. Trénovanie som vypol po 21. epoche z dôvodu, že úspešnosť 22. epochy bola 54,75%. Výsledok 21. epoche je **93,18**%. Túto hodnotu beriem za konečnú

hodnotu trénovania. Priebeh trénovania siete je znázornený na obrázku 5.1. Sieť sa učila približne 17 hodín. Pre každú sieť som vytvoril confusion maticu, ktorá znázorňuje, ako sieť vyhodnotila obrázky oproti správnemu ohodnoteniu obrázkov. Confusion matica je znázornená na obrázku 5.2.



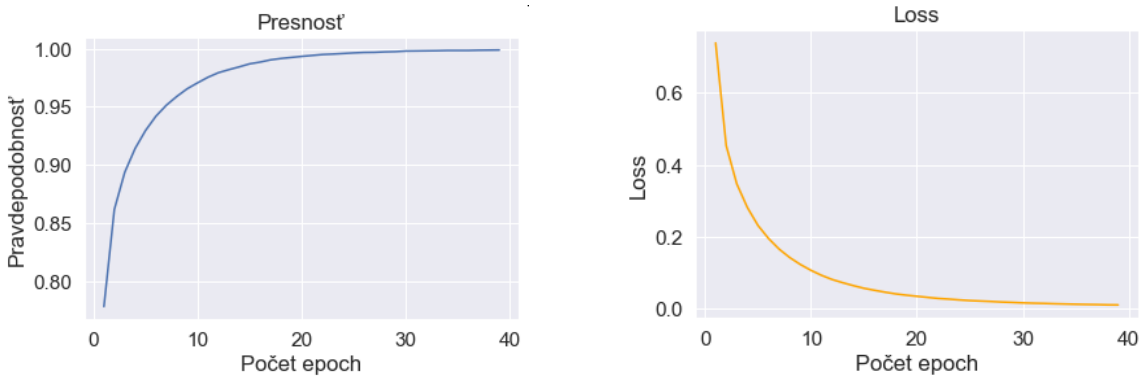
Obr. 5.1: Priebeh trénovania VGG16 - vľavo presnosť, vpravo loss

Obr. 5.2: Confusion matica siete VGG16

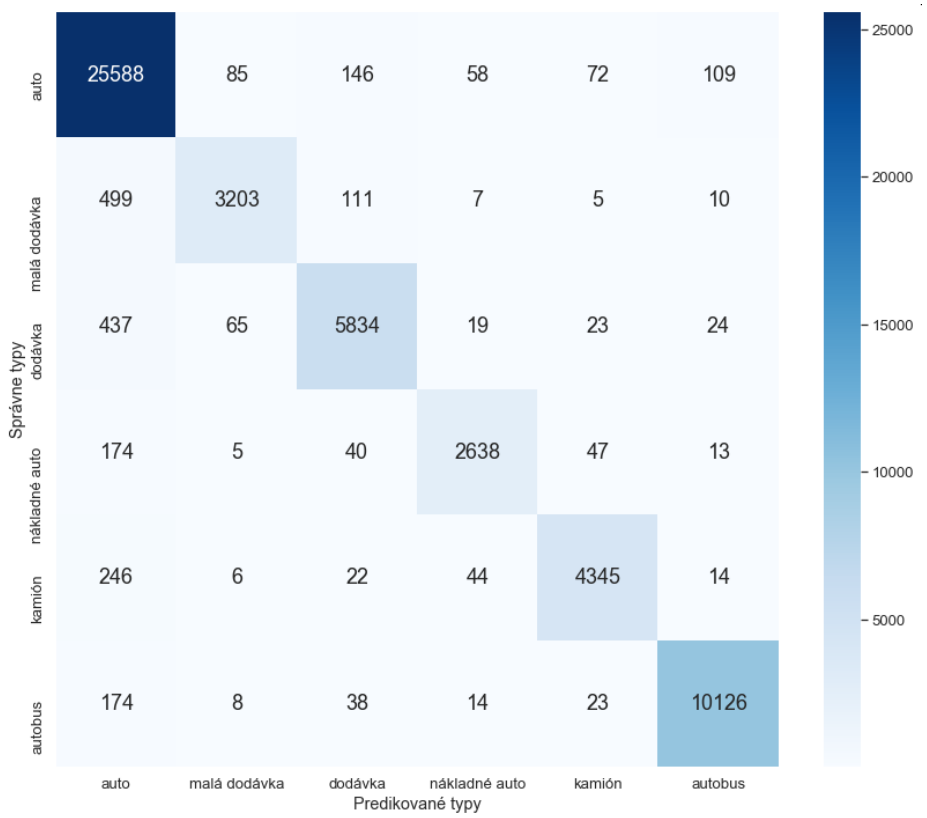
5.5.2 ResNet50

Architektúru ResNet50 som tiež na začiatku nechal trénovať 5 epoch. Po otestovaní na testovacej sade som zistil, že výsledok je horší ako u sieti VGG16. Po 5. epochách sieť dosiahla

úspešnosť iba 49,77%. Nechal som ju však ďalej trénovať. Testovanie som robil pravidelne. Sieť sa na trénovanie chytila, ale mala pomalší štart. Po 12. epoche bola úspešnosť 52,49%. Trénovanie som však ponechal ďalej. Po 20. epoche presnosť bola už na 66,28%. Sieť mala pomalší rozbeh oproti VGG16, ale začala sa učiť. Po 34. epoche bola presnosť 90,76%. Obidve siete berú na vstup obrázok o veľkosti 224×224. Preto veľkosť batch bola tiež nastavená na hodnotu 32. Priebeh trénovania je zobrazený na obrázku 5.3. Sieť rýchlo dosiahla presnosť trénovania na hodnotu 0,99 a loss hodnota bola rýchlo pod úrovňou 0,1. Architektúra sa však stále trénovala s lepšími výsledkami. ResNet50 som nechal trénovať 39 epoch, lebo po tejto epoche sa sieť pretrénovala. Architektúra sa trénovala približne 26 hodín. Výsledná presnosť je **95,35**%. Confusion matica je znázornená na obrázku 5.4.



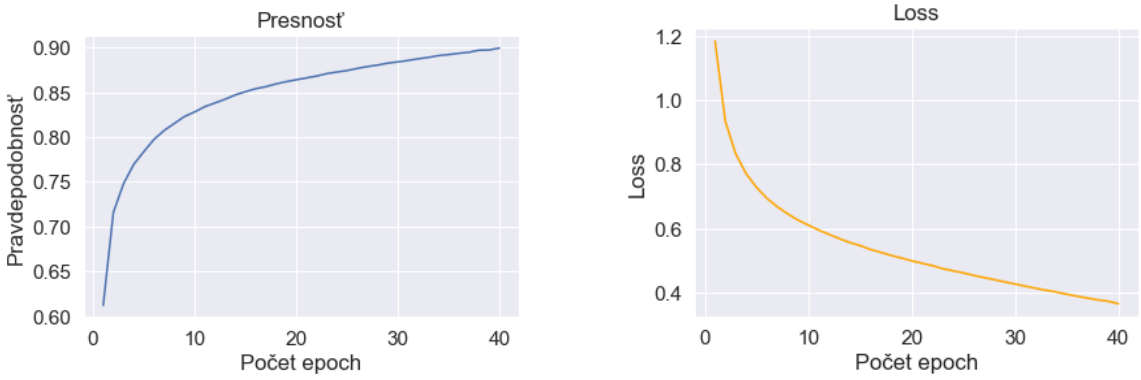
Obr. 5.3: Priebeh trénovania ResNet50 - vľavo presnosť, vpravo loss



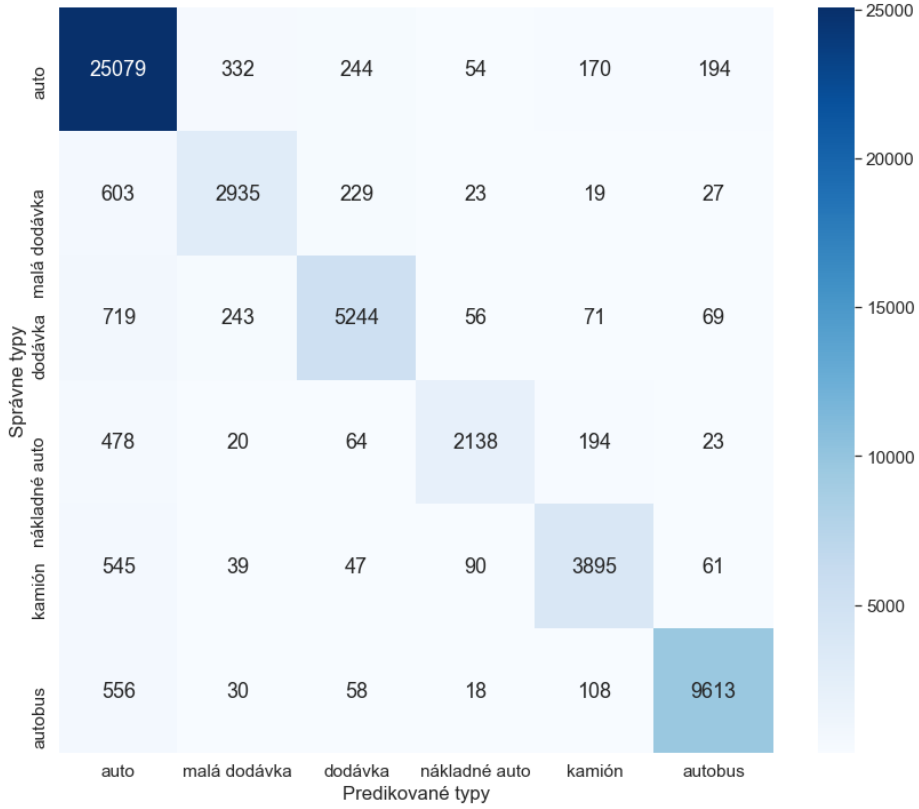
Obr. 5.4: Confusion matica siete ResNet50

5.5.3 Xception

Po trénovaní ResNet50 som začal trénovať Xception. Sieť po prvých epochách vykazovala správne učenie. Preto som túto sieť nechal trénovať dlhšie. Tak isto som pravidelne, po každých 5. epochách, kontroloval výsledky na testovacej sade. Sieť som nechal trénovať 40 epoch. Dôvodom bolo to, že presnosť sa už veľmi málo zlepšovala a trénovanie trvalo dlhšiu dobu. Pre túto sieť, a tiež Inception-ResNet, som musel nechať batch veľkosť na hodnote 16. Z toho dôvodu sa siete dlhšie učili. Xception sa trénovala približne 83 hodín. Priebeh trénovania je znázornený na obrázku 5.5. Výsledok trénovania je **90,08**%. Confusion matica je znázornená na obrázku 5.6.



Obr. 5.5: Priebeh trénovania Xception - vľavo presnosť, vpravo loss

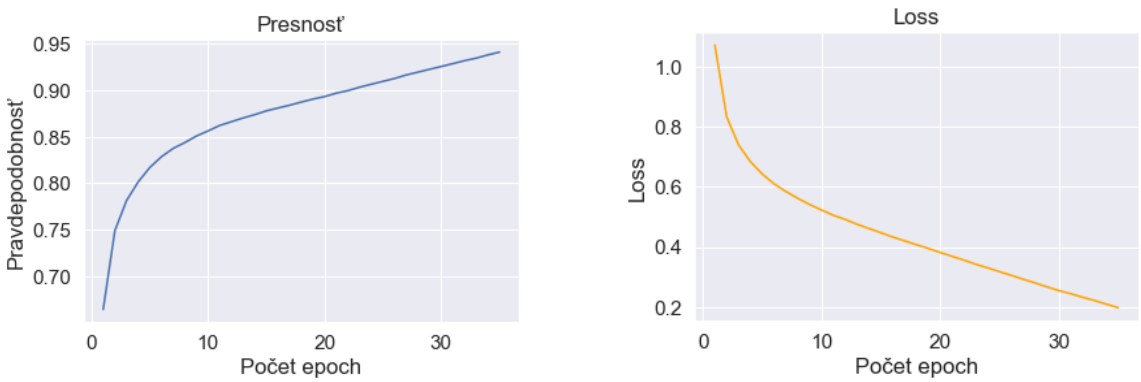


Obr. 5.6: Confusion matica siete Xception

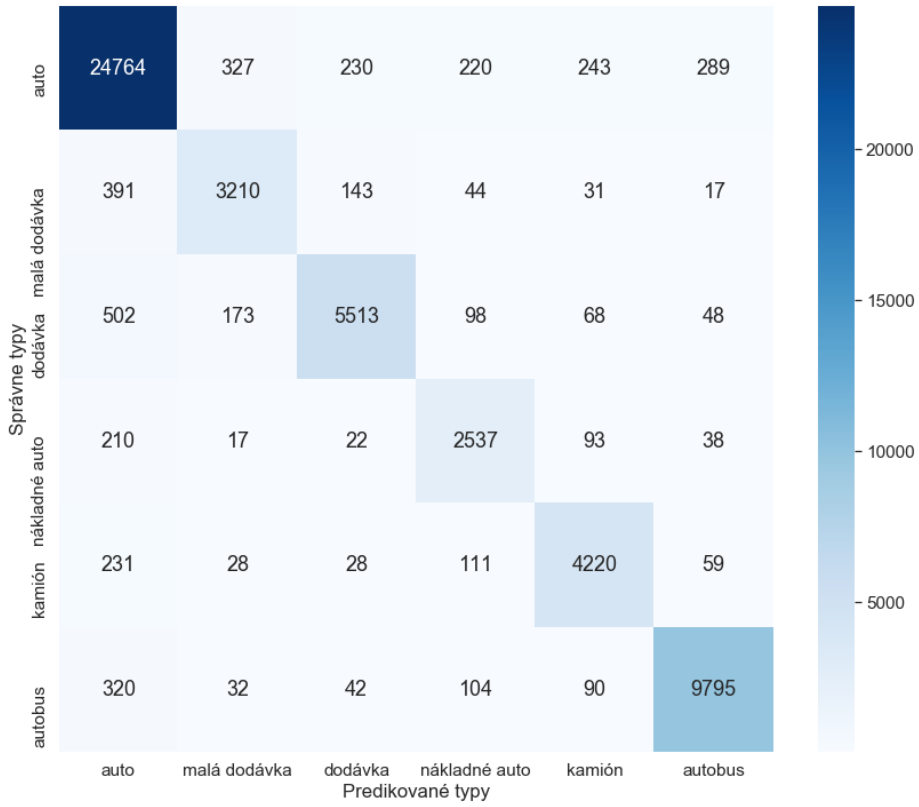
5.5.4 Inception-ResNet-v2

Inception-ResNet-v2 som trénoval rovnako ako Xception. Batch veľkosť bola rovnaká - 16. Architektúra Inception-ResNet-v2 oproti Xception po každom testovaní dosahovala lepšie výsledky. Sieť som trénoval 40 epoch. Za výsledok trénovania beriem prvých 35 epoch. Dôvodom je to, že po 35. epoche, sieť na testovacej sade dávala horšie výsledky. Bol to príznak pretrénovania siete. Inception-ResNet-v2 sa trénovala približne 73 hodín. Výsledok

siete je **92,17**%. Priebeh trénovania je znázornený na obrázku 5.7 a confusion matica je zobrazená na obrázku 5.8.



Obr. 5.7: Priebeh trénovania Inception-ResNet-v2 - vľavo presnosť, vpravo loss



Obr. 5.8: Confusion matica siete Inception-ResNet-v2

5.6 Zhrnutie

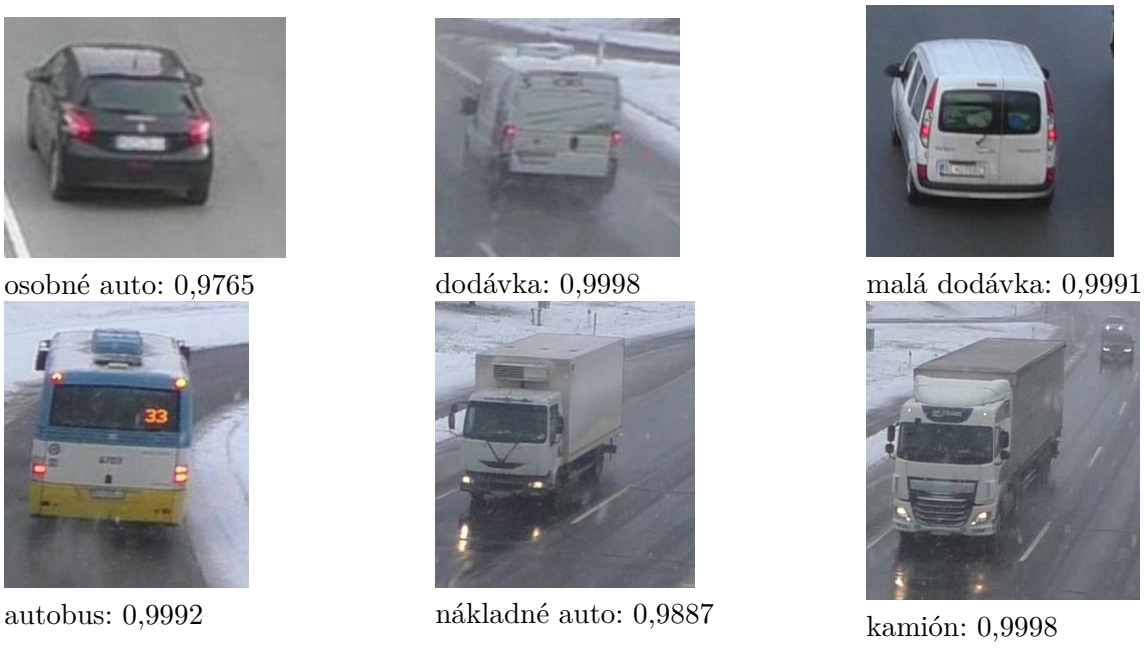
V tejto kapitole bola popísaná implementácia trénovania sietí. V podkapitole 5.1 bol popísaný postup spracovania dát. Podkapitola 5.2 vysvetlila dáta generátor, ktorý slúži na dodávanie obrázkov na trénovanie.

Trénovanie som rozdelil na 3 experimenty. Prvými dvoma experimentami som hľadal úspešné parametre na natrénovanie architektúr. V prvom experimente 5.3 som ponechal celú dátovú sadu. V druhom experimente 5.4 som zredukoval počet osobných áut a zmenil som optimizer. V treťom experimente 5.5 som ponechal dátovú sadu z druhého experimentu a zmenil som *learning rate*.

Tretí experiment považujem za úspešný. Všetky siete sa podarilo natrénovať a vykazujú úspešnosť nad 90%. Architektúra VGG16 sa pretrénovala už po 21. epoche. Architektúra ResNet50 mala naopak veľmi pomalý štart. Myslel som si, že sa sieť s daným optimizerom a s danou *learning rate* nenatrénuje. Po 20. epoche však vykazovala už lepšie hodnoty testovania. Architektúra Xception sa trénovala 40 epoch. Pri posledných epochách sa sieť trénovala veľmi pomaly a tiež klesala loss funkcia. Architektúra Inception-ResNet-v2 sa po 35. epoche pretrénovala a úspešnosť začala klesať. Výsledky trénovania sú znázornené v tabuľke 5.5. Najlepší výsledok má sieť **ResNet50** a to **95,32**%. Sieť sa na danú presnosť trénovala približne 26 hodín. Hlavným dôvodom rýchlosti trénovania bol fakt, že táto sieť pracuje so vstupnými obrázkami o veľkosti 224×224. Preto sa veľkosť batch mohla nastaviť na hodnotu 32. Obrázok 5.9 znázorňuje predikciu architektúry ResNet50 na vybraných obrázkoch z testovacej sady.

	Počet správne určených	**Počet zle určených**	**Úspešnosť**
VGG16	50 571	3 701	**93,18%**
ResNet50	51 734	2 538	**95,32%**
Xception	48 904	5 384	**90,08%**
Inception-ResNet	50 039	4 249	**92,17%**

Tabuľka 5.5: Tabuľka výsledkov architektúr



Obr. 5.9: Obrázok znázorňujúci predikciu ResNet50 na vybraných obrázkoch

Kapitola 6

Záver

Cieľom tejto bakalárskej práci bolo klasifikovať druh vozidla z obrazu pomocou neurónových sietí. Vo svete už existujú funkčné architektúry neurónových sietí, ktoré sa zaoberajú klasifikáciou objektov. Ja som vybral 4 z nich. Natrénoval som ich a porovnal výsledky. Pred začiatkom riešenia práce, som naštudoval teóriu neurónových sietí, existujúce architektúry a nástroje pre tvorbu neurónových sietí.

V rámci práce sa rozpoznávalo 6 typov vozidiel a to: osobné auto, malá dodávka, dodávka, nákladné auto, kamión a autobus. Dataset bol vytvorený z videozáznamu zachytávajúci premávku. Z videí sa pomocou Mask-RCNN vytvorili obrázky trajektórie vozidla. Dataset obsahoval 594 797 obrázkov vozidiel.

Následne na anotovanie všetkých obrázkov som zostrojil anotačný nástroj. Anotačný nástroj bol webovou aplikáciou.

Architektúry, ktoré som vybral na trénovanie boli: VGG16, ResNet50, Xception a Inception-ResNet-v2. V rámci práce som urobil 3 experimenty, kým som siete úspešne natrénoval. V experimentoch som skúšal odlišné parametre optimizera a *learning rate*. Úspešne natrénovať siete sa mi podarilo v treťom experimente, v ktorom som zredukoval dátovú sadu a našiel som správny optimizer s *learning rate*. Najlepší výsledok dosiahla architektúra **ResNet50** s presnosťou **95,32**%. VGG16 dosiahla presnosť 93,18%, Xception dosiahla presnosť 90,08% a Inception-ResNet-v2 dosiahla presnosť 92,17%. Výsledky sú veľmi pozitívne, lebo sa jednalo o náročné dáta.

Prekvapením je to, že staršie architektúry ako VGG16 a ResNet50 dosiahli na tomto datasete lepší výsledok ako novšie architektúry Xception a InceptionResNet-v2. Natrénované siete by mohli byť potencionálne využité na kamerách, ktoré by rozpoznávali typ vozidla na ceste. Siete by tak isto mohli byť použité s detektorom, ktorý detekuje vozidlo z obrazu. Vznikol by nástroj, ktorý by detekoval a rovno klasifikoval typ vozidla.

Literatúra

[1] *Caffe2*. [Online; navštívené 11.01.2020]. Dostupné z: https://research.fb.com/downloads/caffe2/.

[2] *Comparison of AI frameworks*. [Online; navštívené 11.01.2020]. Dostupné z: https://pathmind.com/wiki/comparison-frameworks-dl4j-tensorflow-pytorch.

[3] *Convolutional Neural Networks*. [Online; navštívené 24.01.2019]. Dostupné z: https://cs231n.github.io/convolutional-networks/.

[4] *Convolution and Normalized Cross Correlation on Kepler Architecture* . [Online; navštívené 25.01.2019]. Dostupné z: https://sipl.eelabs.technion.ac.il/wp-content/uploads/sites/6/2016/10/project-image-1599-2-13.png.

[5] *Deep Learning Frameworks Comparison*. [Online; navštívené 11.01.2020]. Dostupné z: https://www.netguru.com/blog/deep-learning-frameworks-comparison.

[6] *Example of CNN architecturen* . [Online; navštívené 25.01.2019]. Dostupné z: https://www.researchgate.net/profile/Takio_Kurita/publication/320748406/figure/fig1/AS:555719381274624@1509505233044/An-example-of-CNN-architecture.png.

[7] *Image-net*. [Online; navštívené 12.01.2020]. Dostupné z: http://www.image-net.org/.

[8] *Inception module*. [Online; navštívené 12.01.2020]. Dostupné z: https://i.stack.imgur.com/zTinD.png.

[9] *Inception module, naive version*. [Online; navštívené 12.01.2020]. Dostupné z: https://nicolovaligi.com/inception_module.png.

[10] *Inteligentné ekonomické systémy*. [Online; navštívené 24.01.2019]. Dostupné z: http://programujte.com/galerie/2005/08/200508191445_obr1.jpg.

[11] *Keras*. [Online; navštívené 11.01.2020]. Dostupné z: https://keras.io/.

[12] *Keras Applications*. [Online; navštívené 25.04.2020]. Dostupné z: https://keras.io/applications/.

[13] *Medical xpress*. [Online; navštívené 24.01.2019]. Dostupné z: https://3c1703fe8d.site.internapcdn.net/newman/gfx/news/hires/2018/2-whyareneuron.jpg.

[14] *Mini-network replacing a 5x5 convolution operation*. [Online; navštívené 13.01.2020]. Dostupné z: https://www.researchgate.net/profile/Giles_Tetteh/publication/316075846/figure/fig1/AS:482670678417408@1492089065033/Left-Mini-network-replacing-a-5x5-convolution-operation-Right-Mininetwork-replacing-a.png.

[15] *Perceptron* . [Online; navštívené 23.01.2019]. Dostupné z: https://cs.wikipedia.org/wiki/Perceptron#/media/File:LinearneSeparovatelne.png.

[16] *ResNet, torchvision, bottlenecks, and layers not as they seem.* [Online; navštívené 14.01.2020]. Dostupné z: https://miro.medium.com/max/1980/1*j_lC2gsO1Kbia8PIQGHUZg.png.

[17] *Specify Layers of Convolutional Neural Network* . [Online; navštívené 13.01.2020]. Dostupné z: https://www.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html.

[18] *What is TensorFlow? Introduction, Architecture Example.* [Online; navštívené 11.01.2020]. Dostupné z: https://www.guru99.com/what-is-tensorflow.html#1.

[19] *What is the highest number of "deep layers"that has been used in any useful neural network?* [Online; navštívené 13.01.2020]. Dostupné z: https://qph.fs.quoracdn.net/main-qimg-854ffd6410140129819823d6c64ac1ca.webp.

[20] *What is vgg?* [Online; navštívené 12.01.2020]. Dostupné z: https://qph.fs.quoracdn.net/main-qimg-e657c195fc2696c7d5fc0b1e3682fde6.

[21] Amidi, A. a Amidi, S. *A detailed example of how to use data generators with Keras.* [Online; navštívené 25.04.2020]. Dostupné z: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly.

[22] Chollet, F. *Xception: Deep Learning with Depthwise Separable Convolutions.* 2017. [Online; navštívené 3.02.2020]. Dostupné z: https://arxiv.org/pdf/1610.02357.pdf.

[23] Deshpande, A. *The 9 Deep Learning Papers You Need To Know About.* 2016. [Online; navštívené 12.01.2020]. Dostupné z: https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html.

[24] Fukushima, K. *Neocognitron.* [Online; navštívené 24.01.2019]. Dostupné z: http://www.scholarpedia.org/article/Neocognitron.

[25] Goodfellowm, I., Bengio, Y. a Courville, A. *Deep learning.* MIT Press, 2016. Dostupné z: http://www.deeplearningbook.org.

[26] He, K., Zhang, X., Ren, S. a Sun, J. *Deep residual learning for image regocnition.* 2015. [Online; navštívené 13.01.2020]. Dostupné z: https://arxiv.org/pdf/1512.03385.pdf.

[27] Iofee, S. a Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015. [Online; navštívené 13.01.2020]. Dostupné z: https://arxiv.org/pdf/1502.03167v3.pdf.

[28] Kharkovyna, O. *Top 10 Best deep learning frameworks in 2019.* [Online; navštívené 11.01.2020]. Dostupné z: https://towardsdatascience.com/top-10-best-deep-learning-frameworks-in-2019-5ccb90ea6de.

[29] Krizhevsky, A., Sutskever, I. a Hinton, G. E. *ImageNet Classification with Deep Convolutional Neural Networks.* 2012. [Online; navštívené 12.01.2020]. Dostupné z: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[30] Kvasnička, V. *Úvod do teórie neurónových sietí.* Iris, 1997. Dostupné z: https://books.google.cz/books?id=0lI6AAAACAAJ. ISBN 9788088778301.

[31] LeCun, Y., Bottou, L., Bengio, Y. a Haffner, P. *Gradient-Based Learning Applied to Document Recognition.* 1998. [Online; navštívené 12.01.2020]. Dostupné z: http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf.

[32] Nagda, R. *Evaluating models using the Top N accuracy metrics.* [Online; navštívené 12.01.2020]. Dostupné z: https://medium.com/nanonets/evaluating-models-using-the-top-n-accuracy-metrics-c0355b36f91b.

[33] Ngiam, J., Chen, Z., Chia, D., Koh, P. W., Le, Q. V. et al. *Tiled convolutional neural networks.* [Online; navštívené 24.01.2019]. Dostupné z: https://papers.nips.cc/paper/4136-tiled-convolutional-neural-networks.pdf.

[34] Novák, M. *Umělé neuronové sítě : teorie a aplikace.* Praha : C.H. Beck, 1998. ISBN 80-7179-132-6.

[35] Řehořek, T. *Umělé neuronové sítě a neuroevoluce.* [Online; navštívené 24.01.2019]. Dostupné z: https://users.fit.cvut.cz/~rehorto2/otevrena-fakulta/neural-networks.html.

[36] Shetty, S. *What is PyTorch and how does it work?* [Online; navštívené 11.01.2020]. Dostupné z: https://hub.packtpub.com/what-is-pytorch-and-how-does-it-work/.

[37] Shi, S., Wang, Q., Xu, P. a Chu, X. *Benchmarking State-of-the-Art Deep Learning Software Tools.* [Online; navštívené 11.01.2020]. Dostupné z: https://arxiv.org/pdf/1608.07249v7.pdf.

[38] Simonyan, K. a Zisserman, A. *Very deep convolutional networks for large-scale image recognition* . 2015. [Online; navštívené 12.01.2020]. Dostupné z: https://arxiv.org/pdf/1409.1556.pdf.

[39] Szegedy, C., Loffe, S., Vanhoucke, V. a Alemi, A. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, url =.*

[40] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. et al. *Going deeper with convolutions* . 2014. [Online; navštívené 12.01.2020]. Dostupné z: https://arxiv.org/pdf/1409.4842.pdf.

[41] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. a Wojna, Z. *Rethinking the Inception architecture for computer vision.* 2015. [Online; navštívené 13.01.2020]. Dostupné z: https://arxiv.org/pdf/1512.00567v3.pdf.

[42] Vondrák, I. *Umělá inteligence a neuronové sítě.* Ostrava : Vysoká škola báňská - Technická univerzita, 2000. ISBN 80-7078-949-2.

[43] Zbořil, F. V. *Základy umělé inteligence.* [Online; navštívené 24.01.2019]. Dostupné z: https://www.fit.vutbr.cz/study/courses/IZU/private/1718izu_9.pdf.

[44] Šíma, J. a Neruda, R. *Teoretické otázky neuronových sítí.* Praha: Matfyzpress, 1996. ISBN 80-85863-18-9.