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 VÝROBCE A MODELU AUTOMOBILU V OBRAZE

VEHICLE MAKE AND MODEL RECOGNITION IN IMAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN BUCHTA**
AUTHOR

VEDOUCÍ PRÁCE **Prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

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

Zadání bakalářské práce

Student: **Buchta Martin**
Program: Informační technologie
Název: **Rozpoznání výrobce a modelu automobilu v obraze**
Vehicle Make and Model Recognition in Image
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou analýzy dopravy pomocí videa. Zaměřte se na detekci vozidel a jejich klasifikaci.
2. Vyhledejte a prostudujte dostupné přístupy k rozpoznání modelu automobilu v obraze.
3. Vyhledejte dostupné datové sady pro učení a vyhodnocování systémů pro rozpoznání modelu automobilu v obraze. Pořiďte a anotujte vlastní dílčí datovou sadu.
4. Experimentujte s vhodnými přístupy k rozpoznání modelu automobilu v obraze.
5. Vylaďte vybraný přístup k maximální přesnosti a zvažte možnosti jeho urychlování.
6. Vyhodnoťte vytvořenou metodu na vhodných datech a diskutujte výsledky.
7. Zhodnoťte vlastnosti vytvořeného řešení a navrhněte možnosti pokračování; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Jakub Sochor, Jakub Špaňhel, Adam Herout: BoxCars: Improving Fine-Grained Recognition of Vehicles Using 3-D Bounding Boxes in Traffic Surveillance, IEEE Transactions on Intelligent Traffic Systems, 2018
- Jakub Sochor, Adam Herout, Jiří Havel: BoxCars: 3D Boxes as CNN Input for Improved Fine-Grained Vehicle Recognition, IEEE Conference on Computer Vision and Pattern Recognition, 2016
- J. Yang, B. Price, S. Cohen, H. Lee, M.-H. Yang: Object Contour Detection with a Fully Convolutional Encoder-Decoder Network, IEEE Conference on Computer Vision and Pattern Recognition 2016
- J. Krause, M. Stark, J. Deng, L. Fei-Fei: 3D Object Representations for Fine-Grained Categorization, CVPR Workshops 2013
- N. Gählert, J.-J. Wan, M. Weber, J. M. Zöllner, U. Franke, J. Denzler: Beyond Bounding Boxes: Using Bounding Shapes for Real-Time 3D Vehicle Detection from Monocular RGB Images, IEEE Intelligent Vehicles Symposium (IV) 2019

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, značné rozpracování bodů 4 a 5.

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

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 5. listopadu 2019

Abstrakt

Tato práce se věnuje problematice klasifikace modelu automobilu z obrazu. Popisuje několik metod, jako konvoluční neuronové sítě, metody omezené na přední/zadní pohled a metody využívající 3D CAD modely. Z těchto přístupů si vybírá konvoluční neuronové sítě, kterým se dále věnuje. Práce obsahuje popis jednotlivých vrstev, ze kterých se taková síť skládá. V praktické části je popsán postup, jakým byl vytvořen klasifikátor, který má přesnost 80,7 %. Pro účely ověření funkčnosti byla vytvořena datová sada obsahující 1 034 fotografií. Práce dále experimentuje s různými architekturami a vyhodnocuje jejich přesnost. Součástí práce je program, který díky detektoru automobilu najde ve videu vozidla a v daném videu je označí čtverečkem a popisem modelem automobilu.

Abstract

This thesis deals with classification of a car model from an image. It describes several methods, such as convolutional neural networks, methods limited to the fron/rear view and methods using 3D CAD models. From these approaches it chooses convolutional neural networks, which it further deals with. The work contains a description of the individual layers of which such a network consists. The practical part describes the procedure by which the classifier, that has an accuracy of 80.7 %, was created. A dataset containing 1 034 photos was created to verify functionality. The work further experiments with different architectures and evaluates their accuracy. The work contains a program which, thanks to the car detector, finds the vehicle in the video and marks it with a square and a description of the car model in the given video.

Klíčová slova

konvoluční neuronové sítě, klasifikace modelu automobilu, klasifikace obrazu, detekce automobilu v obraze, strojové učení, umělá inteligence

Keywords

convolutional neural networks, vehicle model classification, image classification, vehicle detection in the image, machine learning, artificial intelligence

Citace

BUCHTA, Martin. *Rozpoznání výrobce a modelu automobilu v obraze*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Adam Herout, Ph.D.

Rozpoznání výrobce a modelu automobilu v obraze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Prof. Ing. Adama Herouta Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Buchta
28. května 2020

Poděkování

Tímto bych chtěl poděkovat panu Prof. Ing. Adamu Heroutovi Ph.D. za jeho čas a odborné rady při psaní této bakalářské práce. Taktéž děkuji svým rodičům, rodině a kamarádům za jejich podporu během celého studia.

Tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvořit klasifikátor, který na základě fotografie automobilu určí, o který model automobilu se jedná. Takový klasifikátor se dá použít např. ke zjišťování statistik o vozovém parku, nebo může být používán bezpečnostními složkami k automatické kontrole zákazu vjezdu různým typům vozidel (např. SUV do center měst nebo kamionům na dálnice). Tato práce se zaměřuje na fotografie z dohledové kamery, které jsou dnes běžně umístěny na mýtných branách nebo na stožárech lamp. Pro využití tohoto řešení tedy nebude nutné instalovat dodatečné kamery nebo jiná zařízení, je plně funkční při využití dnešní infrastruktury a s běžnými výpočetními kapacitami. Výsledný klasifikátor by měl rozlišovat mezi výrobcem a modelem automobilu.

Musel jsem se vypořádat s hledáním existujících přístupů ke klasifikaci modelu automobilu. Potřeboval jsem najít takový způsob, který bude umět rozlišovat mezi co největším množstvím tříd s co největší přesností. Ruku v ruce s hledáním vhodných přístupů jsem hledal vhodnou datovou sadu, která by obsahovala fotografie aut, jež se podobají pohledu z dohledových kamer na stožárech podél silnic. Dalším kritériem pro hledanou datovou sadu bylo, že by měla vzniknout v Evropě, protože složení vozových parků v Evropě a například v USA je značně odlišné. Volbu přístupu ke klasifikaci a volbu datové sady jsem nevnímal jako dva samostatné procesy, nýbrž pro každou metodu jsem hledal vhodné datové sady. Po nalezení vhodné datové sady a vybrání přístupu, který použiji, jsem hledal technologie, na kterých tento klasifikátor postavím. Použité technologie jsem vybíral tak, aby byly dostatečně vysokoúrovňové a aby mi, jakožto vývojáři, co nejvíce usnadnily práci. Po vybrání technologií jsem hledal prostředky, na kterých tyto technologie spustím. Trénování klasifikátoru je obecně výpočetně náročné, obzvlášť pro obrazová data. Nejprve jsem zkoušel trénování na běžném notebooku, poté jsem přidal akceleraci pomocí grafické karty. Vyzkoušel jsem cloudové výpočetní zdroje komerčních firem a nakonec i národní gridovou infrastrukturu – Metacentrum. Po vybrání přístupu, datové sady i technologií jsem klasifikátor navrhl a natrénoval. Navrhl jsem vlastní architektury, ale taky jsem přebíral architektury již existující. Provedením mnoha experimentů jsem se snažil dosáhnout co nejlepší úspěšnosti mého řešení. Povedlo se mi dosáhnout úspěšnosti 80,7 %.

V rámci této práce jsem vytvořil vlastní dílčí datovou sadu, která obsahuje 1 034 fotografií automobilů z 36 různých tříd. Tato datová sada je vhodná pro ověření schopnosti generalizace klasifikátoru. Mé řešení mělo na této vlastní datové sadě Top-5 úspěšnost 93,5 %. Data jsem získával z videozáznamu, který jsem pořídil v Brně. Fotografie automobilů jsem získal automaticky díky detektoru vozidel. Jednotlivým fotografiím jsem následně ručně přiřadil třídu.

První dvě kapitoly této práce se věnují teoretické části. Zbývající kapitoly popisují mé řešení a vyhodnocení úspěšnosti. V kapitole 2 rozebírám existující přístupy k problému klasifikace modelu automobilu z obrazu. Uvádím tři možnosti: konvoluční neuronové sítě, metody omezené na přední/zadní pohled a metody založené na 3D CAD modelech. Ve zbytku této práce se zaměřím na práci s konvolučními neuronovými sítěmi. V kapitole 3 popisuji jednotlivé části, ze kterých se taková síť skládá. Kapitola 4 obsahuje popis, jak jsem postupoval při tvorbě a trénování mého klasifikátoru. V této kapitole se mimo jiné zabývám použitými technologiemi, které jsem si pro tvorbu klasifikátoru vybral (Python, Tensorflow, OpenCV...). Tvorbou vlastní datové sady se zabývám v kapitole 5. V této kapitole se zabývám také vyhodnocením úspěšnosti mého výsledného klasifikátoru na vlastní datové sadě. Na závěr v kapitole 6 provádím experimenty, během kterých se zaměřuji na architekturu konvoluční neuronové sítě a úpravy trénovacích dat.

Kapitola 2

Existující přístupy k rozpoznání modelu automobilu v obraze

V následující kapitole budou nastíněny známé možnosti, jak se s problémem klasifikace modelu automobilu můžeme vypořádat. Pod pojmem klasifikace modelu automobilu myslíme rozpoznání výrobce, modelu a generace (roku výroby) daného automobilu. Tato kombinace je také někdy označována jako „jemnozrnná“ klasifikace.

Úkol jemnozrnné klasifikace modelu automobilu se oproti jiným klasifikačním úlohám může opřít o to, že všechny automobily mají určité společné vlastnosti. Některé metody se spoléhají na detekci registrační značky a klasifikaci objektů kolem ní, jiné metody se spoléhají na světlomety a názarníkovou oblast. Na druhou stranu přístupy nezávislé na úhlu pohledu se nemohou spolehnout na detekci těchto částí automobilu.

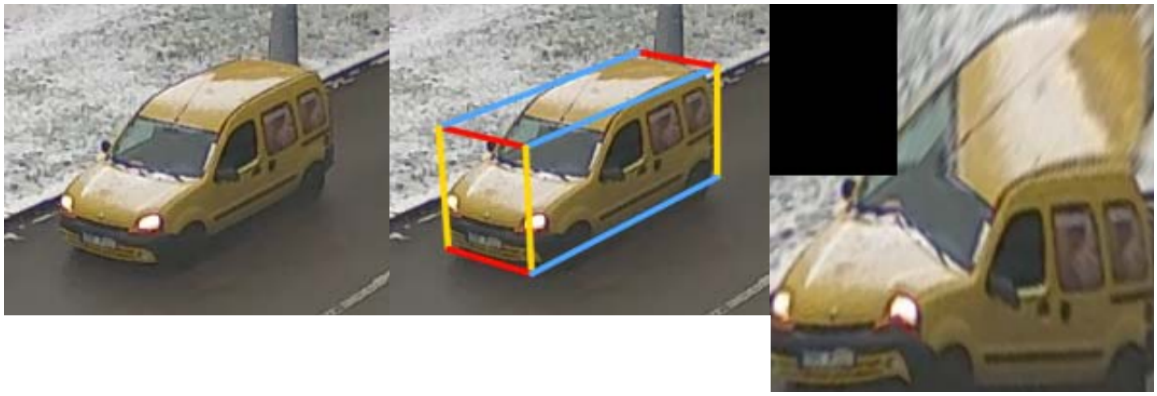
Problémem při klasifikaci modelu automobilu je to, že jsou si mnohé modely automobilů vizuálně velmi podobné.

2.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou systémy, na jejichž vstupu je obrázek a na výstupu je predikce třídy. V porovnání s tradičními způsoby, které jsou závislé na ručně navržené extrakci příznaků, konvoluční neuronové sítě provádí extrakci příznaků automaticky.

Práce *Fine-Grained Vehicle Model Recognition Using A Coarse-to-Fine Convolutional Neural Network Architecture* [12] ke klasifikaci modelu automobilu používá konvoluční neuronové sítě a SVM klasifikátor [11]. Tento model je omezen na čelní pohled. Nejprve se na vstupním obrázku identifikují části pomocí konvoluční neuronové sítě. Další dvě konvoluční sítě slouží ke klasifikaci světlometů a nárazníkové zóny. Výstupy těchto tří neuronových sítí slouží jako vstupy do SVM klasifikátoru, který tvoří finální predikci. Autoři této metody dosáhli úspěšnosti 98,29 % při rozlišování 281 tříd automobilů.

Metoda popsaná v článcích [34] a [36] ke klasifikaci využívá konvoluční neuronové sítě. Vylepšením tohoto postupu je úprava fotografie před samotným vstupem do neuronové sítě, kdy se provádí operace *unbox*. Nejprve se kolem automobilu sestaví 3D kvádr (*bounding box*), tyto boxy umožní identifikovat přední/zadní (**F**), boční (**S**) a střešní stěnu (**R**) vozidla. Stěny tohoto 3D bounding boxu jsou v operaci unbox použity k normalizaci fotografie automobilu, během které se tyto stěny „rozbalí“ do roviny (**U**). Horní levá podmatice je vyplněna nulami. Ukázka této metody je na obrázku 2.1.



Obrázek 2.1: Příklad vstupní fotografie (vlevo), 3D bounding box (uprostřed) a normalizovaného obrázku po provedení unbox metody (vpravo). Převzato z [34].

$$\mathbf{U} = \begin{pmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{F} & \mathbf{S} \end{pmatrix} \tag{2.1}$$

Konstrukce 3D bounding boxu okolo automobilu na původní fotografii je kritické pro správné fungování této metody. Práce *Automatic Camera Calibration for Traffic Understanding* [10] přináší způsob, jak tyto bounding boxy automaticky získat na základě informace o konturách vozidla.

Ve výše zmíněných článcích [34, 36] autoři dosáhli zlepšení úspěšnosti klasifikace u určitých architektur konvolučních neuronových sítí až o 12 procentních bodů.

2.2 Metody omezené na přední/zadní pohled

Metoda popsaná v článku *Multi-class Vehicle Type Recognition System* [8] je omezená na fotografie přední nebo zadní strany automobilu. Principem této metody je detekce poznávací značky automobilu a extrakce příznaků v okolí kolem poznávací značky. Tyto příznaky (jako světla, nárazník, obrysy výdechu chladiče atd. . .) se ukázaly jako dostatečně diskriminativní.

Tato metoda pracuje pouze s černobílými obrázky. Každý model automobilu má svůj prototyp (obrázek přední/zadní strany). Všechny prototypy mají předem daný rozměr, v tomto případě 600×252 pixelů. Na trénovacím obrázku se nejprve detekuje poznávací značka. Pozice značky je popsána čtveřicí bodů (rohů značky) $\{A, B, C, D\}$. Poté je na obrázek aplikována transformace, po které budou rohy značky na požadovaných bodech $\{A', B', C', D'\}$. Příklad prototypu je na obrázku 2.2. Na tento prototyp se použijí Sobelovy filtry k získání směru a velikosti gradientu v každém bodě. Pro klasifikaci modelu automobilu je použitý algoritmus nejbližšího souseda.

Přistup popsaný v článku *Vehicle model recognition using geometry and appearance of car emblems from rear view images* [26] je omezen na pohled na zadní stranu automobilu. Metoda nejprve detekuje poznávací značku automobilu a normalizuje fotografii pomocí transformace popsané výše. Postup se skládá ze tří kroků. V každém kroku se výběr možných tříd omezuje, až nakonec v posledním kroku je vybrána jedna třída. Nejprve se klasifikuje výrobce automobilu podle loga, poté je určen model automobilu omezený výbě-

Obrázek 2.2: Příklad vstupní fotografie a jeho prototypu. Převzato z článku *Multi-class Vehicle Type Recognition System* [8].

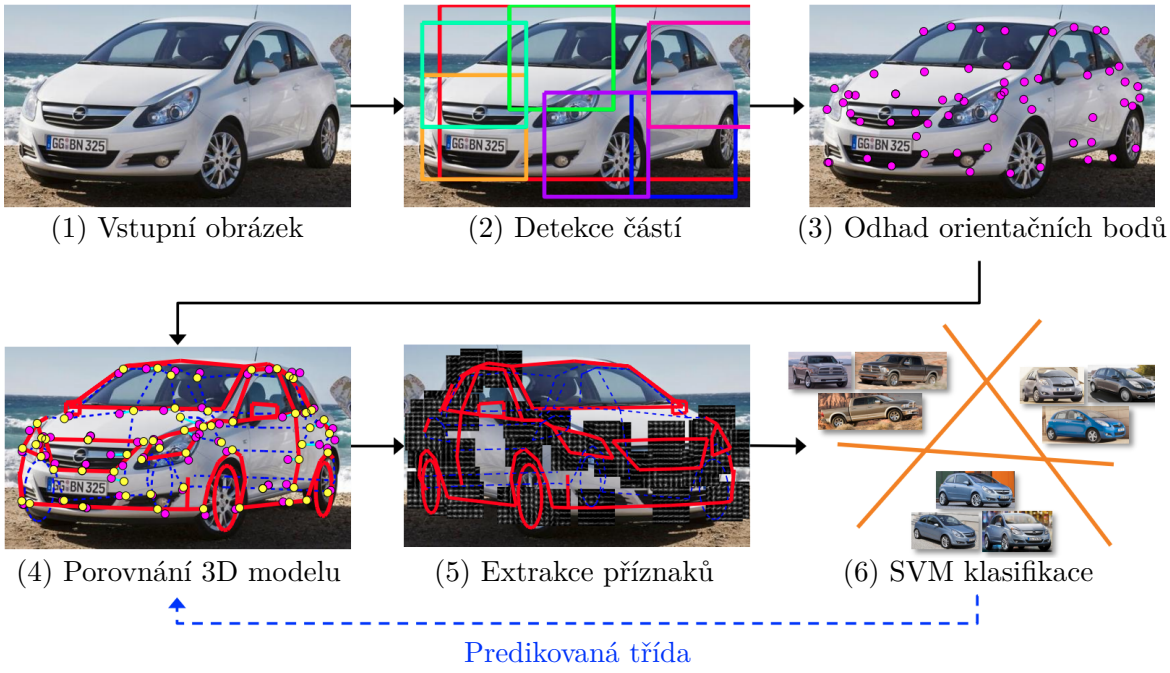
rem modelů daného výrobce, nakonec se určí generace (rok výroby) podle geometrie. Tato metoda používá k extrakci příznaků histogram orientovaných gradientů (HOG) [9] a ke klasifikaci používá lineární SVM [11] klasifikátor. Autoři dosáhli úspěšnosti 94 % na datasetu obsahujícím 8 výrobců, 28 modelů a 52 různých generací (tříd).

2.3 Metody založené na 3D CAD modelech

Lin a kolektiv ve své práci *Jointly Optimizing 3D Model Fitting and Fine-Grained Classification* [24] navrhli iterativní algoritmus skládající se z následujících kroků (viz obrázek 2.3):

1. Nový obrázek na vstupu
2. Detekce částí automobilu pomocí DPM (Deformable part model) [13]
3. Odhad orientačních bodů v obraze vytvořený předtrénovaným regresním modelem na základě odhadnutých částí z bodu 2, inicializace 3D modelu predikované třídy náhodným 3D modelem
4. Porovnání orientačních bodů z 3D modelu s orientačními body z obrazu
5. Extrakce příznaků s využitím histogramu orientovaných gradientů (HOG) [9] a Fisher vektoru [28]
6. Nový odhad třídy podle Support Vector Machine (SVM) [11], načtení 3D modelu nově predikované třídy, skok na bod 4, dokud model nekonverguje

Tato metoda je použitelná pro fotografie pořízené z libovolného úhlu, což je podstatná změna oproti metodám založených na detekci poznávací značky automobilu (viz kapitola 2.2). Problémem této metody je náročné vytváření datasetu, který je potřeba k vytvoření dostatečně robustního modelu. Dataset vytvořený pro tento výzkum (*FG3DCar*) [24] obsahuje 300 obrázků ze 30 tříd. Na každém obrázku bylo ručně vyznačeno 64 orientačních bodů. Pro každou třídu musí být dostupný odpovídající CAD model. Vytváření datasetu pro konvoluční neuronovou síť (viz kategorie 3) je snazší, protože stačí zařadit fotografii do jedné z tříd.



Obrázek 2.3: Princip iterativního modelu založeném na 3D CAD modelech. Převzato z článku *Jointly Optimizing 3D Model Fitting and Fine-Grained Classification* [24].

Kapitola 3

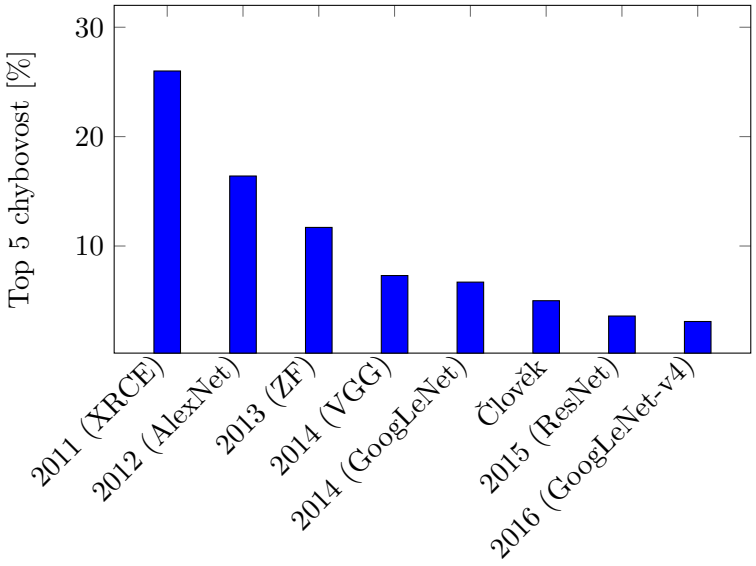
Konvoluční neuronové sítě

Pro klasifikaci obrazu jsou vhodné konvoluční neuronové sítě [27]. Projekt ImageNet [21] vytvořil databázi pro klasifikaci obrázků do 22 000 různých tříd. Databáze ImageNet[1] obsahuje více než 14 milionů lidmi kategorizovaných obrázků, které jsou reálné a ve vysokém rozlišení (průměrně 482×418 px[2]). Od roku 2010 se každoročně koná soutěž The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [30], které se pravidelně zúčastňují zástupci významných vědeckých i průmyslových institucí. Jedním z problémů, které se na ILSVRC řeší, je klasifikace obrazu. Všechny týmy dostanou určitou podmnožinu ImageNet databáze, na které trénují své modely. Každým rokem se úspěšnost vítězného modelu zlepšuje a dnes je přesnost těchto modelů dokonce lepší než přesnost člověka [17]. V roce 2012 byl v klasifikaci obrazu poprvé nejlepší model, který používal konvoluční neuronové sítě. Tento model – AlexNet [18] – i vítězové v letech následujících (jako VGG [33], GoogLeNet [39], ResNet [19] a další...) používají konvoluční neuronové sítě. Úspěšnost vybraných modelů spolu s úspěšností člověka (5 %) je zobrazena na obrázku 3.1, který zobrazuje *Top-5 chybovost*, což je pravděpodobnost, že daný model nezařadil správnou třídu mezi 5 nepravděpodobnějších tříd. Pro vyjádření pravděpodobnosti, že model nedal správné třídě nejvyšší skóre, se používá pojem *Top-1 chybovost*.

V konvolučních neuronových sítích se vstupní vektor posílá do *konvoluční vrstvy*, kde probíhá 2D konvoluce. Výsledek konvoluce se po aplikaci *aktivační funkce* (typicky ReLu) dále posílá do tzv. *pooling vrstvy*, kde se snižuje velikost dimenzí. Bloky tvořené konvoluční vrstvou, aktivační funkcí a pooling vrstvou se typicky několikrát opakují. Výsledek poslední konvoluční vrstvy se převede do 1D vektoru (*Flatten vrstva*) a tato data jsou vstupem do *plně propojené vrstvy* následované aktivační funkcí. Plně propojených vrstev spolu s aktivační funkcí může být více. Poslední plně propojená vrstva je následovaná aktivační funkcí Sigmoid a má takový počet neuronů, kolik existuje různých tříd v klasifikační úloze. Detailní popis jednotlivých vrstev najdete níže v této kapitole.

V kapitole 4 se budeme zabývat právě použitím konvolučních neuronových sítí pro úlohu klasifikace modelu automobilu v obraze. Budeme postupovat obdobně, jako kdybychom chtěli natrénovat síť na dataset Imagenet.

[1] http://www.image-net.org
[2] https://devopedia.org/imagenet



Obrázek 3.1: Chybovost vybraných konvolučních sítí. Zdroj dat: edge-ai-vision.com [5]

3.1 Konvoluce pro počítačové vidění

Konvoluční vrstva slouží k vytvoření mapy příznaků (feature map). Konvoluce přibližuje princip klasifikace obrazu tomu, jak lidský mozek klasifikuje obraz. [15] Člověk většinu dat zahodí a pracuje pouze s příznaky. (Při identifikaci obličeje to může být nos, ústa, oko a podobně...) Matematický zápis 2D konvoluce najdeme v rovnici 3.1, kde I je výška a J je šířka konvolučního jádra.

$$y[k,l] = y[k,l] * h[k,l] = \sum_{m=\frac{I-1}{2}}^{\frac{I-1}{2}} \sum_{n=\frac{J-1}{2}}^{\frac{J-1}{2}} h[m,n]x[k-m,l-n] \tag{3.1}$$

$$H = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{3.2}$$

Příkladem matice konvolučního jádra je matice H z rovnice 3.2. Tato matice je tzv. Sobelův operátor a slouží jako detektor horizontálních hran. Příklad aplikace Sobelových operátorů je na obrázku 3.2. Úpravou vah v konvolučním jádru můžeme získat detektor hran v libovolném směru. Pokud se posuneme hlouběji do konvoluční neuronové sítě, následující vrstvy se mohou aktivovat pouze, pokud detekují určitý vzor. Např. určitá množina hran, která tvoří daný vzor. Tyto vzory pak jsou vstupem pro další vrstvy neuronové sítě, které z těchto vzorů mohou tvořit komplexnější vzory, nebo rovnou celé objekty. Takto se tvoří příznaky, které jsou poté vstupem do plně propojené vrstvy. V kontextu konvolučních neuronových sítí se někdy o konvolučním jádru hovoří jako o *detektoru příznaků*.

V konvolučních neuronových sítích se v konvolučních vrstvách používá vícero konvolučních jader. Výstupem této vrstvy je několik příznakových map (pro každé konvoluční jádro jedna). Během procesu trénování (vit kapitola 3.6) se konvoluční jádra mění podle potřeb. Běžným příkladem rozměru konvolučního jádra používaným v konvolučních neuronových sítích je filtr o rozměru $3 \times 3 \times 3$, který představuje 3×3 filtr přes všechny 3 barevné

Obrázek 3.2: Lenna a aplikace detektoru horizontálních hran (uprostřed) a vertikálních hran (vpravo). Převzato z článku *Introduction to Convolutional Neural Networks* [43].

složky najednou. Konvoluční jádra s takovým rozměrem můžeme najít třeba v síti ResNet [19], která obsahuje desítky takovýchto konvolučních vrstev. Dalšími běžnými rozměry jsou 5×5 a 7×7. S konvolučními jádry větších rozměrů se v konvolučních neuronových sítích setkáváme výjimečně. Konvoluční vrstvy typicky bývají následované aktivační funkcí *ReLU* [25] kvůli potlačení linearity.

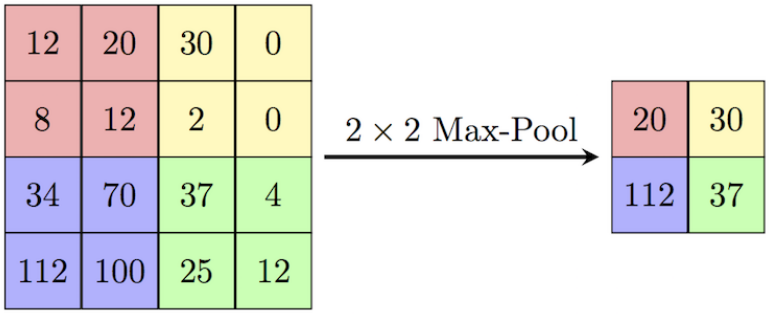
Konvoluční vrstva může taky snižovat rozměry dat. Pokud nastavíme krok (strides) vyšší než 1, výrazně zredukujeme rozměry. Pro krok 2 se výška i šířka sníží na polovinu. Této vlastnosti se ale spíše využívá u následující pooling vrstvy.

3.2 Pooling vrstva

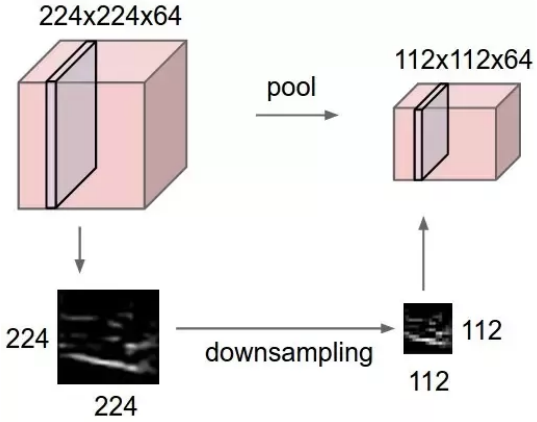
Pooling vrstva (někdy též subsamplingová vrstva či sdružovací vrstva) v konvolučních neuronových sítích zajišťuje invarianci vůči posunu a do velké míry je zodpovědná za schopnost generalizovat při klasifikačních úlohách. Pooling vrstva redukuje množství parametrů proudících v síti, což je vidět na obrázku 3.4.

Na vstupu pooling vrstvy je výstup předchozí vrstvy (zpravidla konvoluční vrstvy – příznaková mapa). Pooling při každém kroku pracuje pouze s odpovídající podmaticí příznakové mapy, z ní vybere maximální hodnotu a tuto hodnotu pak uloží na odpovídající místo na výstupní matici mapy sdružených příznaků. Poolingu existuje více druhů. V praxi se nejvíce používá *max pooling*, který vybírá maximální hodnotu z dané podmatice. Dalším typem je *average pooling*, který počítá aritmetrický průměr hodnot v dané podmatici. *Sum pooling* vyrobí z dané podmatice součet všech hodnot (sumu). Názorný příklad max poolingu je na obrázku 3.3.

Velikost masky je často 2×2 (např. pro síť ResNet [19]). Dalším parametrem této vrsty je krok (strides), se kterým se tato maska „posouvá“ po vstupní matici. Výhodou pooling vrstvy je, že zachová informaci o výskytu příznaků, které nás zajímají. Tento výskyt příznaků je ve výstupní matici reprezentován vysokými hodnotami. Zároveň se použitím pooling vrstvy zbavíme většiny informací o příznacích, které nehledáme, což je dobrá prevence proti přetrénování.



Obrázek 3.3: Příklad max poolingu. Převzato z computersciencewiki.org [2].

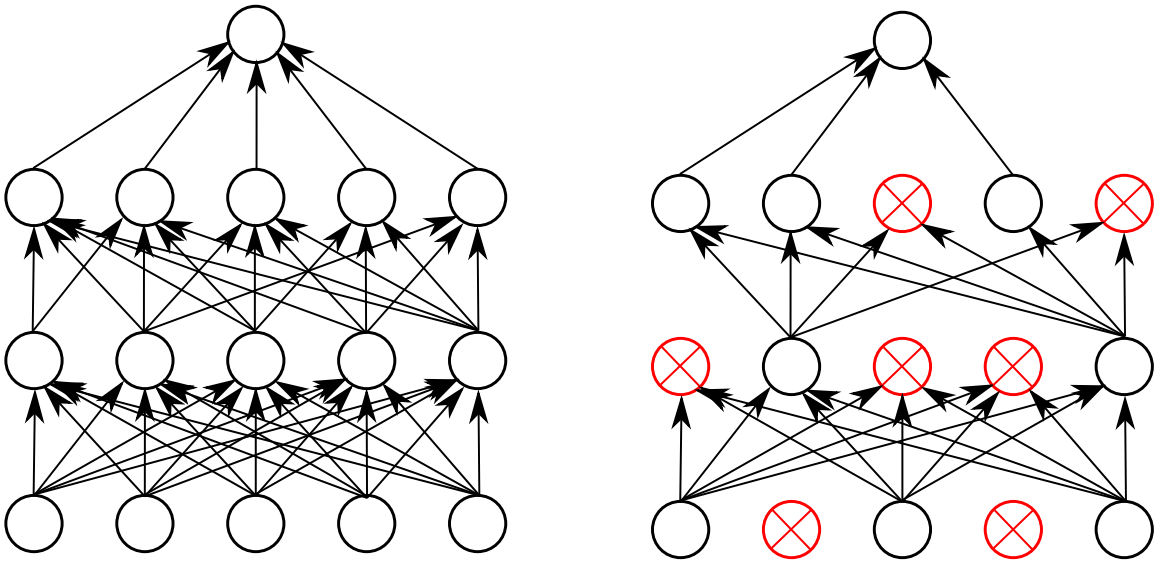


Obrázek 3.4: Ukázka zmenšení toku dat v síti díky pooling vrstvě. Převzato z computersciencewiki.org [2].

3.3 Plně propojená vrstva

Po poslední pooling vrstvě je nutné data transformovat tak, aby mohla být použita jako vstup do plně propojené vrstvy. Toto nám zajistí tzv. *Flatten vrstva* (zplošťující vrstva), která hodnoty z příznakových map umístí do jednoho vektoru.

Výstup flatten vrstvy jde do klasické *neuronové sítě*. Ta se skládá ze vstupní vrstvy, plně propojené vrstvy (někdy tzv. skrytá vrstva) a výstupní vrstvy. Schéma neuronové sítě je na obrázku 3.5. Úkolem těchto vrstev je vytvořit predikci třídy na základě informací o příznacích, které byly zjištěny v konvolučních vrstvách. Každá vrstva se skládá z předem daného počtu neuronů. Každý neuron je propojený se všemi neurony v předchozí vrstvě. Výstupní vrstva má tolik neuronů, kolik tříd požadujeme klasifikovat. Za výstupní vrstvou bývá ještě aktivační funkce *softmax*, která zaručí, že součet výstupů z těchto neuronů bude roven 1 a tyto výstupy tedy můžeme brát jako pravděpodobnosti. Výslednou hodnotu pak bereme jako pravděpodobnost, že obrázek na vstupu odpovídá právě té třídě, kterou daný neuron reprezentuje. Za ostatními plně propojenými vrstvami bývá aktivační funkce *ReLU* [25] kvůli potlačení linearity.



Obrázek 3.5: Ukázka plně propojené vrstvy (vlevo) a stejná síť po aplikaci dropout (vpravo)

3.4 Dropout

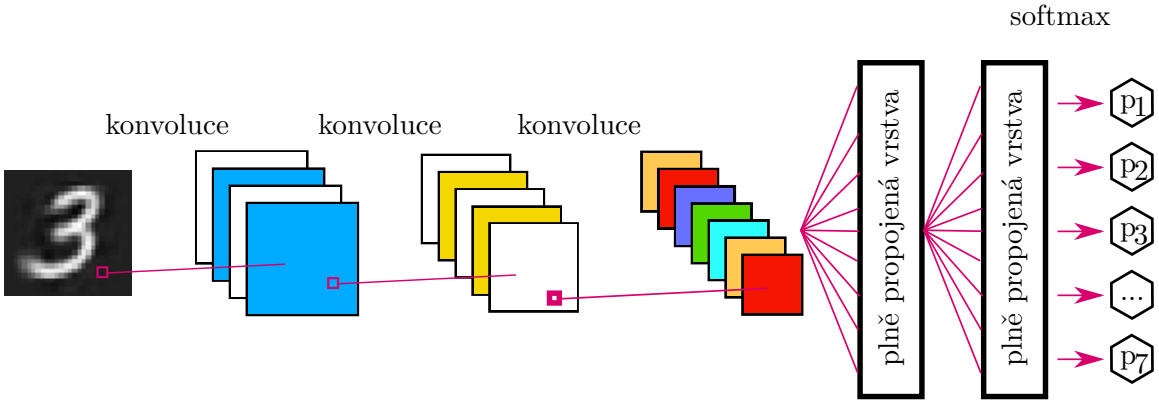
Dropout vrstva je aktivní při trénování sítě a jejím účelem je ignorovat určitá data s předem danou pravděpodobností. Na výstupu dropout vrstvy jsou stejné informace jako na jejím vstupu, ale při každém stádiu trénování se vyberou informace, které se „zakryjí“ pro následující vrstvu. Ukázku použití dropout vrstvy můžeme vidět na schématu z obrázku 3.5 (vpravo). Dropout vrstva se vkládá mezi jednotlivé plně propojené vrstvy, ale v konvolučních neuronových sítích se často vkládá také za pooling vrstvu, kde tvoří šum v obraze.

Smyslem použití dropout vrstvy [37] je zabránění přetrénování (*overfitting*). Pokud se při trénování použije dropout, tak neuronová síť má tendenci rozpoznat více robustní příznaky, které jsou užitečné při použití s mnoha náhodně vybranými podmnožinami zbylých neuronů. Použití dropout vrstvy může až zvojnásobit počet iterací potřebných ke konvergenci. Na druhou stranu proces trénování je pro jednotlivé etapy rychlejší, než při použití všech neuronů z předchozí vrstvy [7]. Dropout vrstva má jeden parametr a tím je pravděpodobnost, se kterou dojde k „zakrytí“ daného neuronu.

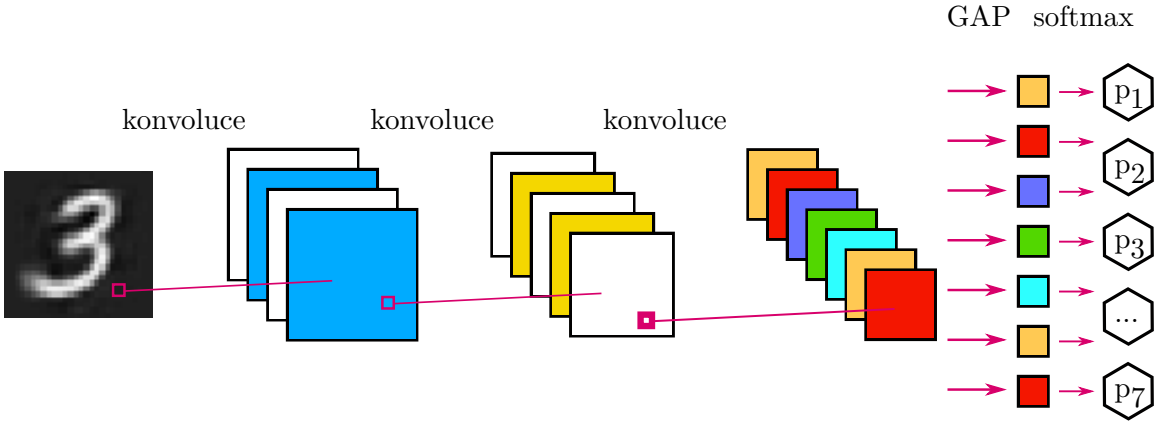
3.5 Global average pooling vrstva

Global average pooling vrstva [23] (zkráceně někdy GAP) je vrstva použitá v architekturách GoogLeNet a jiných z ní vycházejících (viz kapitola 4.4.2). Jedná se o vrstvu, která se nachází místo plně propojených vrstev (tzn. za poslední konvoluční vrstvou nebo za poslední pooling vrstvou). Nevýhodou plně propojených vrstev na konci konvenčních konvolučních neuronových sítí je, že mají příliš mnoho parametrů, je výpočetně (a časově) náročné je natrénovat a navíc mají tendenci způsobovat přetrénování.

Hlavní myšlenkou použití global average pooling vrstvy je vytvoření jedné mapy příznaků pro každou třídu. V konvenčních architekturách (jako VGG16, ResNet, AlexNet...) je výstup poslední konvoluční vrstvy (vektor map příznaků) vstupem do plně propojených vrstev. Global average pooling ale každou mapu příznaků zprůměruje, z každé mapy pří-



Obrázek 3.6: Ukázka architektury konvoluční neuronové sítě s využitím plně propojené vrstvy (bez využití global average pooling vrstvy)

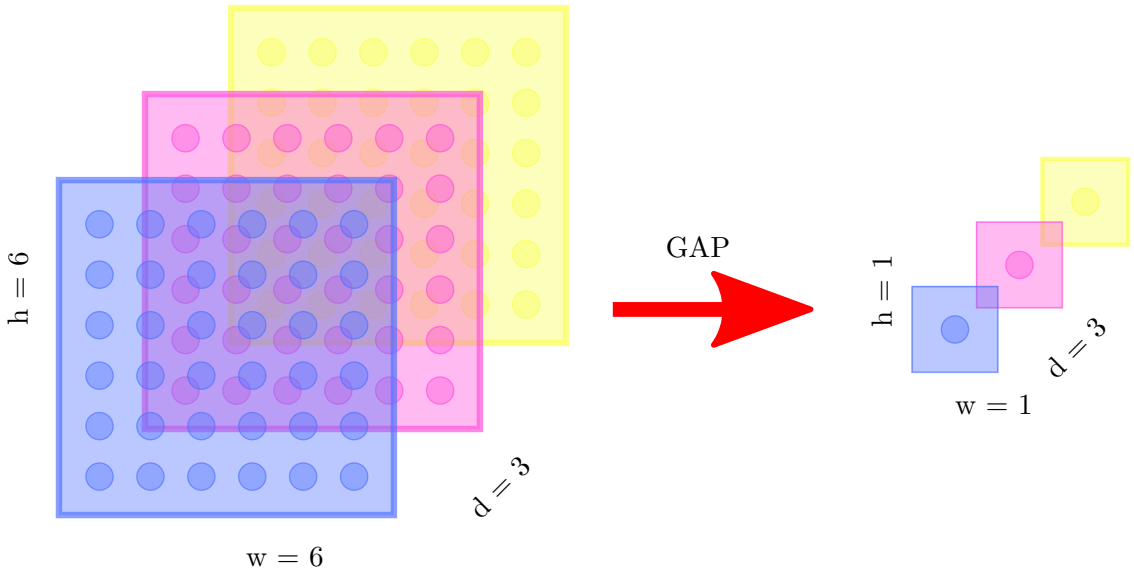


Obrázek 3.7: Ukázka architektury konvoluční neuronové sítě s využitím global average pooling vrstvy (bez využití plně propojené vrstvy)

znaků vznikne jeden skalár. Tyto sklaláry jsou seřazeny do vektoru a tento vektor průměrů příznakových map slouží jako vstup do vrstvy, která aplikuje aktivační funkci softmax.

Výhodou použití global average pooling vrsty je, že tato vrstva nemá žádný parametr, který by se optimalizoval. Oproti tomu plně propojené vrstvy na konci konvenčních architektur mají podstatný počet parametrů. Např. v klasické VGG 16 architekruře (která používá plně propojené vrstvy na konci, viz kapitola 4.4.2) tvoří parametry plně propojených vrstev $89,4\%$ všech parametrů. Tím, že snížíme počet parametrů sítě, zlepšíme její schopnost generalizace a snížíme náchylnost k přetrénování. Architektura ResNet k tomuto problému přistupuje kompromisně tak, že používá global average pooling vrstvu a za ní je jedna plně propojená vrstva, která má pro každou třídu jeden perceptron s aktivační funkcí softmax.

Na obrázku 3.6 najdete schéma architektury konvoluční neuronové sítě, která používá plně propojenou vrstvu pro porovnání s obrázkem 3.7, který znázorňuje síť, která používá global average pooling vrstvu. Na obrázku 3.8 je znázorněno fungování global average pooling vrstvy.



Obrázek 3.8: Schéma fungování global average pooling vrstvy

3.6 Trénování

Pod pojmem trénování neuronové sítě myslíme proces, při kterém se upravují parametry neuronové sítě (váhy, biasy) tak, abychom minimalizovali ztrátovou funkci *loss function* na daném trénovacím datasetu, který se skládá z trénovacích fotografií a jím odpovídajících označení tříd (label). Trénování neuronové sítě je iterativní proces, který se zkládá z *dopředné propagace* (forwardpropagation) a *zpětné propagace* (backpropagation) .

Dopředná propagace je proces, při kterém jsou neuronové síti předložena nová data. Jednotlivé neurony berou informaci z neuronů v předchozí vrstvě, aplikují svou transformaci spolu s aktivační funkcí a posílají data do neuronů v další vrstvě. Jakmile se data dostanou až do poslední vrstvy, tak v této poslední vrstvě vznikne predikce jejich třídy. [40]

K vyčíslení toho, jak dobré nebo špatné jsou predikce neuronové sítě, se používá ztrátová funkce. Tato funkce měří to, jak „vzdálené“ jsou predikované třídy od tříd skutečných. Tyto skutečné třídy známe, protože se jedná o učení s učitelem a skutečné třídy jsou součástí trénovacího datasetu. Trénování neuronové sítě je optimalizační úloha a naším cílem je, aby hodnota ztrátové funkce byla rovna 0 (tzn. aby nebyl rozdíl mezi predikovanými a skutečnými třídami).

Po spočítání ztrátové funkce se hodnota této funkce propaguje zpátky do jednotlivých neuronů. Směrem od výstupní vrstvy, přes skryté vrstvy, až do vstupní vrstvy. Do různých neuronů ve skrytých vrstvách se dostane různá hodnota ztrátové funkce podle toho, jak velký podíl na původní ztrátové funkci měl daný neuron. Takto se informace o chybě dostane skrz všechny vrstvy. Každý neuron má informaci o tom, jak velkým dílem se podílel na původní hodnotě ztrátové funkce.

Nyní, když každý perceptron (model neuronu) má informaci o svém podílu na hodnotě ztrátové funkce, nastává čas pro algoritmus zvaný *gradient descent* [29]. Tento optimalizační algoritmus mění parametry v neuronové síti malými kroky opačným směrem, než je gradient ztrátové funkce. V tomto směru se očekává globální minimum ztrátové funkce. Tato úprava parametrů probíhá pro každou dávku dat (batch) podaný neuronové síti při trénování. Při

každé dávce se směr gradientu (a tudíž i posunu argumentů sítě) může měnit. Toto se děje iterativně pro všechny dávky dat ve všech iteracích.

Trénování probíhá na trénovacích datech. Pro ověření úspěšnosti na neviděných datech slouží validační data. Validační data jsou typicky části datasetu, která neslouží pro trénování sítě, ale po každé iteraci se pošlou k vyhodnocení. Úspěšnosti na těchto neviděných datech se říká validační úspěšnost. V ideálním případě trénovací i validační úspěšnost rostou v každé iteraci a jejich hodnoty jsou podobné. Reálně ale často dochází k přetrénování, které se vyznačuje tím, že testovací úspěšnost je znatelně lepší než validační úspěšnost. Přetrénování je známkou toho, že model nedokáže dobře generalizovat. Zabránit přetrénování se dá např. zjednodušením modelu, rozšířením trénovacích dat, použitím data augmentation metod (viz kapitola 4.5) nebo snížením počtu iterací na počet, kdy ještě nedochází k znatelnému přetrénování.

Kapitola 4

Návrh řešení pro klasifikaci modelu automobilu

Cílem této práce je implementovat klasifikátor pro klasifikaci modelu automobilu. Tento systém na vstupu dostane fotografii automobilu (pořízenou z libovolného úhlu) a na výstupu určí výrobce automobilu a jeho model.

Tato práce je zaměřena na klasifikátory implementované pomocí konvolučních neuronových sítí v programovacím jazyce Python 3. Pro tvorbu těchto neuronových sítí budeme používat knihovnu *Tensorflow*, která zpřístupňuje vysokoúrovňové rozhraní. Výpočty budeme provádět ve výpočetních centrech *Google Colab* a *Metacentrum*.

4.1 Datová sada

Mít vhodnou datovou sadu je klíčové pro úlohy v oblasti strojového učení. Existuje několik datových sad, které jsou určené pro detekci vozidel, nebo pro klasifikaci modelu automobilu pomocí různých metod (viz kapitola 2). Např. dataset *FG3DCar* [24] obsahuje 300 fotografií automobilů, které jsou patřičně označeny typem automobilu pro jemnozrnnou klasifikaci. Tento dataset je ale vytvořený pro klasifikaci na základě 3D CAD modelů. Obsahuje mimo jiné ručně označené orientační body, které jsou použity při klasifikaci s využitím 3D CAD modelů. Dataset vytvořený ke článku *Exploiting effects of parts in fine-grained categorization of vehicles* [22] je zaměřený na čelní pohled na automobil, což je k trénování klasifikátoru schopného klasifikace na fotografiích z libovolného úhlu nevhodné.

Dataset *Cars*[1], který byl vytvořený k článku *3D Object Representations for Fine-Grained Categorization* [20], obsahuje přes 16 tisíc fotografií automobilů z různých úhlů. Yang [44] zveřejnil dataset *CompCars*[2], který obsahuje přes 136 tisíc fotografií rozdělených do 1 700 tříd.

Dataset *BoxCars116k* [35] je zaměřený na pohled z dopravních kamer, které mohou být umístěny např. na lampách nebo mýtných branách. Tento pohled je specifický vyšší výškou kamery a relativně malým rozlišením automobilu. Fotografie zachycují automobily z různých úhlů pohledu (zepředu, zboku i zezadu). Fotografie byly pořízeny z videozáznamu více než 100 dopravních kamer v Brně a okolí. Zachyceno na videu bylo přes 27 tisíc vozidel. Dataset obsahuje 116 tisíc fotografií rozdělených do 693 tříd. Ukázku fotografií můžete vidět na obrázku 4.1.

[1] http://ai.stanford.edu/~jkrause/cars/car_dataset.html
[2] http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html

Obrázek 4.1: Série náhodně vybraných fotografií z datasetu BoxCars116k

Pro účely trénování neuronové sítě byla data rozdělena do množiny trénovacích dat a validačních dat. Každá z 137 kamer byla náhodně přidělena do množiny trénovací nebo validační. Do trénovací množiny se kamera přidala s pravděpodobností 60 %, do validační množiny s pravděpodobností 40 %. Třídy byly vytvořeny tak, abychom byli schopni určit výrobce a model vozidla. Příkladem takovéto třídy je *Škoda Fabia*. Vyřazeny byly ty třídy, které měly v trénovacích datech méně než 15 fotografií, nebo ve validačních datech neměly ani jednu fotografii. Celkem zbylo 99 tříd. V trénovacích datech bylo 68 775 fotografií a ve validačních datech bylo 24 388 fotografií.

4.2 Použité technologie

4.2.1 Python

Python[3] je programovací jazyk, který má široké možnosti využití. Je to multiplatformní skriptovací jazyk s vysokou mírou abstrakce. Je silně dynamicky typovaný. Pro strojové učení v Pythonu existuje velké množství knihoven, které usnadňují práci programátorovi (např. Keras, Pytorch, Scikit-Neural Network...). Klasifikátor popsaný v této práci je implementován v jazyce Python 3.6.

4.2.2 Tensorflow

Tensorflow[4] je open source knihovna pro strojové učení. Tato knihovna využívá systém grafů datových toků, kde uzly daného grafu představují jednotlivé matematické operace, zatímco hrany představují data (tenzory). Tato data „proudí“ (flow) mezi danými operacemi. Knihovna Tensorflow byla původně vyvíjena společností Google, ta ji však v roce 2015 vydala veřejně jako open source projekt. Na Tensorflow je třeba nahlížet jako na výpočetní software, který má zveřejněné API rozhraní pro různé programovací jazyky. Oficiální API

[3] https://www.python.org
[4] https://www.tensorflow.org

```
model = Sequential([
    Conv2D(16, 3, padding='same', activation='relu', input_shape=(64, 64 ,3)),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1)
])

model.compile(
    optimizer='adam',
    loss=BinaryCrossentropy(),
    metrics=['accuracy'],
)
```

Výpis 4.1: Příklad použití knihovny Tensorflow pro vytvoření konvoluční neuronové sítě. Převzato z tensorflow.org [1].

rozhraní existují pro Python a jazyk C. Bez zpětné kompatibility existují rozhraní např. pro C++, Java, Javascript a mnoho dalších... Tensorflow podporuje distribuované učení na více jádrech procesoru zároveň. Pokud máme k dispozici grafickou kartu s architekturou CUDA (grafické karty Nvidia), tak nám Tensorflow umožňuje snadno akcelerovat výpočty na GPU. Google také vyvíjí speciální koprocesory TPU (Tensor processing unit), které jsou specializované na akceleraci strojového učení. Tyto TPU koprocesory jsou veřejně dostupné online na platformě Google Colab (viz kapitola 4.3.2). Knihovna Tensorflow podporuje akceleraci pomocí TPU koprocesorů.

V ukázce kódu 4.1 je příklad použití knihovny Tensorflow pro vytvoření konvoluční neuronové sítě, která na vstupu přijímá obrázky o rozměru 64×64 pixelů s hloubkou 3 kanálů. Na tento vstupní obrázek je aplikována 2D konvoluce (viz kapitola 3.1). Při této konvoluci je použito 16 konvolučních jader. Šířka i výška konvolučních jader je 3. Krok je nastaven na 1. Na závěr je v této vrstvě aplikována aktivační funkce *ReLU*. V další vrstvě probíhá max pooling (viz kapitola 3.2). Použitá velikost mapy pro max pooling je 2×2. Krok je nastaven na 2, takže se výška i šířka výstupních dat sníží na polovinu. Předchozí kombinace konvoluční vrstvy a max pooling vrstvy se opakuje ještě 2x s tím rozdílem, že se v každé další konvoluční vrstvě zdvojnásobí počet konvolučních jader. Za poslední max pooling vrstvou se nachází Flatten vrsta, která vstupní data umístí do jednoho vektoru. Po ní následuje plně propojená vrstva (viz kapitola 3.3) s 512 neurony a aktivační funkcí *ReLU*. Poté následuje poslední vrstva, ve které je jeden neuron, který udává výsledné skóre.

Tento model je vhodný pro binární klasifikaci, protože ve výstupní vrstvě je jeden perceptron. Ve výsledném klasifikátoru použijeme pro každou třídu jeden perceptron v poslední plně propojené vrstvě, který bude udávat pravděpodobnost dané třídy.

Knihovna Tensorflow mimo jiné usnadňuje implementaci *data augmentation* (viz kapitola 4.5). Doslova jedním řádkem kódu se implementují techniky jako horizontální/vertikální

```
image_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
)
```

Výpis 4.2: Příklad použití knihovny Tensorflow pro aplikaci vybraných data augmentation technik. Převzato z tensorflow.org [3].

převrácení, náhodná změna saturace, náhodná změna jasu, náhodná rotace nebo náhodné oříznutí obrazu.

V ukázce kódu 4.2 je příklad použití vybraných technik data augmentation. Použití tohoto objektu třídy *ImageDataGenerator* umožní načítání trénovacích fotografií a rovnou aplikaci náhodných transformací na ně. V tomto případě se jedná o normalizaci dat na hodnoty z intervalu $< 0; 1 >$, náhodnou rotaci obrazu až o 40°, náhodný posun obrazu až o 20 %, zkosení až o 20 %, zoom obrazu až o 20 % a náhodné horizontální převrácení. Více informací o data augmentation najdete v kapitole 4.5.

4.2.3 NumPy

NumPy je knihovna pro jazyk Python, která implementuje práci s vektory, maticemi a vícerozměrnými poli. Veškerá data, se kterými pracuji, jsou NumPy pole. Toto usnadňuje například implementaci vlastních metod data augmentation, knihovna Tensorflow navíc podporuje práci s NumPy.

4.2.4 OpenCV

OpenCV je knihovna pro usnadnění práce s počítačovou grafikou. Jedná se o open-source knihovnu, která byla původně vyvíjena společností Intel. Knihovna OpenCV je implementována v jazyce C++, ale poskytuje rozhraní pro Python i Octave.

4.3 Výpočetní hardware

4.3.1 Vlastní hardware

V rané fázi vývoje klasifikátoru jsem používal vlastní notebook ke trénování neuronové sítě. Notebook má procesor *Intel Core i5 7200U Kaby Lake (2,5 GHz)* a grafickou kartu *NVIDIA GeForce GTX 950M*. Později jsem vlastní výpočetní hardware používal pouze pro ověření funkčnosti a vizualizaci trénovacích dat před spuštěním na vzdálených výpočetních clusterech. Výhodou používání vlastního hardware je, že se nemusí čekat na spuštění úlohy, jak je tomu na Metacentru (viz kapitola 4.3.3), ale úloha se spustí okamžitě. Další výhodou je, že data zůstávají uchována na disku a velikost disku v principu není omezena, což byl problém u Google Colab (viz kapitola 4.3.2).

4.3.2 Google Colab

Google Colab[5] je online služba od společnosti Google pro vývojáře. Služba je zdarma a umožňuje akceleraci výpočtů pomocí grafických karet *Tesla K80 GPU*. [14] Google Colab spouští Python kód, který je ve napsán ve formátu *Jupyter Notebook* (soubory *.ipynb*). Tento formát souboru umožňuje elegantně kombinovat kusy Python kódů a textových poznámek.

Kód je spouštěn na virtuálním stroji, který má kapacitu disku 108 GB a kapacitu RAM paměti 13 GB. K dispozici jsou předem nainstalované knihovny jako Keras, Tensorflow, PyTorch nebo OpenCV. V případě potřeby je možno spustit bash příkazy a např. pomocí příkazu *pip* nainstalovat další knihovny.

Google Colab poskytuje služby zdarma a je určený především pro studenty. Aby se zabránilo nezamýšlenému použití této služby, tak každý Google účet může mít najednou spuštěný maximálně jeden virtuální stroj, uživatel musí být aktivní, maximální doba spuštění každého virtuálního stroje je 12 hodin a data uložená na disku nejsou perzistentní. Pro trénování klasifikátoru to znamená, že je třeba pokaždé stáhnout a rozbalit datovou sadu. Na druhou stranu je Google Colab velmi intuitivní a všechny ovladače a verze knihoven i Pythonu jsou předem vyladěné.

4.3.3 Metacentrum

Ve fázi, kdy jsem měl připravenou datovou sadu a měl jsem vyzkoušenou práci s knihovnami na zmenšených fotografiích, jsem začal využívat zejména výpočetní výkon poskytovaný Metacentrem[6]. Nejprve je nutné vyplnit přihlášku (a prokázat příslušnost k akademické instituci v České republice) a počkat na její schválení. Poté se na výpočetní klastry přihlašuje přes SSH.

Na Metacentru existují dva hlavní typy úložišť:

- **Diskové pole** se nachází v `/storage`. Tento prostor funguje jako primární úložiště souborů. Toto úložiště je trvalé a soubory zde uložené jsou uloženy permanentně. V tomto úložišti se nacházejí domovské adresáře, je to vhodné úložiště pro datové sady a skripty. Pro diskové pole je důležitá především kapacita.

- **Scratch úložiště** se nachází v `/scratch`. Toto úložiště bývá uloženo na rychlých SSD discích. Scratch úložiště slouží výhradně jako úložiště pro dočasné soubory během výpočtů. Každá úloha má vlastní scratch úložiště a jeho velikost je daná požadavky uživatele při vkládání nové úlohy. Po spuštění úlohy je dobrým zvykem si zkopírovat datovou sadu z diskového pole do scratch úložiště, protože načítání jednotlivých souborů je výrazně rychlejší, což má podstatný vliv na celkovou rychlost. Před ukončením úlohy je vhodné si výsledky uložit na diskové pole, protože výsledky zůstanou perzistentní. Po skončení úlohy se obsah scratch úložiště smaže.

Plánování a spouštění jednotlivých úloh má na starosti software *Portable Batch System* (PBS). Tento systém má dva základní typy úloh:

- **Interaktivní úloha** umožňuje alokaci potřebných zdrojů (počet CPU, GPU, velikost RAM, velikost scratch úložiště...) a zbytek je na uživateli, který úlohu řídí prostřednictvím interaktivního shellu. Interaktivní úlohy jsou vhodné pro lazení úloh

[5] https://colab.research.google.com
[6] https://www.metacentrum.cz

a hledání chyb. Pro běžné spouštění výpočtů ale moc vhodné nejsou, protože uživatel musí čekat, dokud jeho úloha nebude spuštěna a poté se musí čekat na vstup uživatele. Zároveň je třeba mít po celou dobu běhu úlohy spuštěný terminál s SSH připojením. Pokud se připojení přeruší, celý běh úlohy se zastaví a výpočetní zdroje budou uživateli odebrány.

- **Dávková úloha** je reprezentována jako batch script. Jedná se o klasický shell script, který obsahuje informace pro PBS interpret, ve kterých jsou uvedeny nároky na potřebné zdroje, název úlohy a jiné informace. Po spuštění úlohy se vykoná obsah tohoto scriptu. Výhodou dávkových úloh je možnost automatizace a možnost běhu na pozadí. Na druhou stranu se v dávkových úlohách hůře hledá chyba a nejsou vhodné pro ladění. Příklad scriptu pro dávkovou úlohu najdete v ukázce kódu 4.3.

```
#!/bin/bash
#PBS -N davkovaUloha
#PBS -l walltime=23:00:00
#PBS -q gpu
#PBS -l select=1:ncpus=1:ngpus=1:mem=16gb:scratch_local=64gb
```

Výpis 4.3: Příklad scriptu pro dávkovou úlohu v PBS

4.4 Architektura konvoluční neuronové sítě

4.4.1 Vlastní architektury

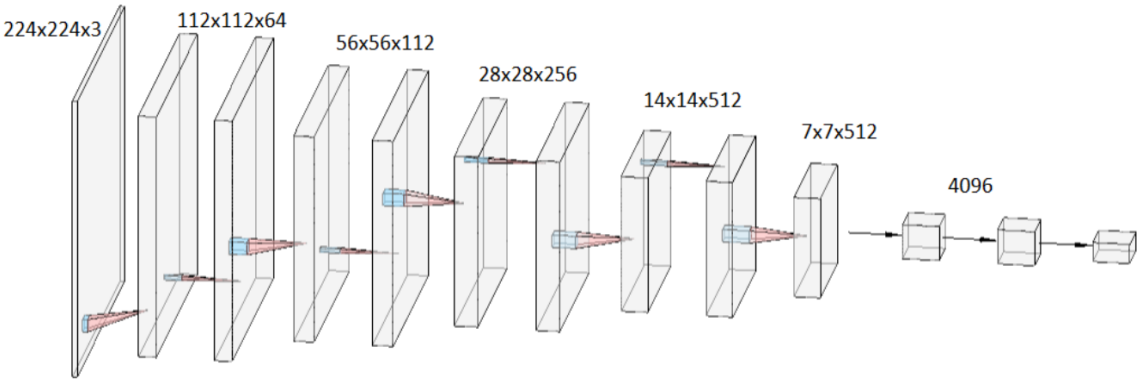
Z počátku jsem se snažil vytvořit vlastní architekturu konvoluční neuronové sítě. Pracoval jsem s jednotlivými vrstvami, optimalizoval jejich počet, pořadí a parametry. Používal jsem sekvenční model neuronové sítě, což znamená, že výstup předchozí vrstvy slouží jako vstup do vrstvy bezprostředně následující.

Vlastní architektury se skládají z bloků tvořených konvoluční vrstvou, pooling vrstvou a dropout vrstvou. Konvoluční vrstvy jsou následované aktivační funkcí *ReLU*. Popis jednotlivých vrstev najdete v kapitole 3. V některých případech byla pooling vrstva v každém sudém bloku vynechána. Těchto bloků bylo za sebou několik (typicky 4 až 10). Po posledním bloku následovaly 1 až 2 plně propojené vrstvy s aktivační funkcí *ReLU*. Poslední plně propojená vrstva byla následovaná aktivační funkcí *Sigmoid* a měla 133 perceptronů, protože dataset obsahuje 133 různých tříd (viz kapitola 4.1).

Největší úspěšnost dosahovala architektura na obrázku 4.2. Jedná se o architekturu, která obsahuje 9 výše zmíněných bloků. Každý druhý blok nemá pooling vrstvu. Po posledním bloku jsou dvě plně propojené vrstvy, každá z nich má 4 096 perceptronů. Následuje plně propojená vrstva se 133 perceptrony. Tato architektura dosahuje Top-1 úspěšnosti 56 % na validačních datech. Těchto výsledků bylo dosaženo bez data augmentation metod.

4.4.2 Převzaté architektury

Protože úspěšnost na vlastních architekturách nebyla dostačující, přistoupil jsem k převzetí cizích architektur konvolučních neuronových sítí. Parametry těchto převzatých architektur neuronových sítí byly inicializovány hodnotami, které pochází z trénování těchto architektur



Obrázek 4.2: Schéma vlastní architektury konvoluční neuronové sítě

na datovou sadu ImageNet popsanou v kapitole 3. Při trénování se tyto parametry upravují tak, aby neuronová síť dobře klasifikovala modely automobilů, ale tvary křivek běžně se vyskytujících v přírodě již neuronová síť zná, na rozdíl od situace, kdy jsou parametry sítě inicializovány náhodně. Experimentování s těmito neuronovými sítěmi probíhalo převážně v nastavení parametrů plně propojených vrstev, které za převzatými architekturami následují a v použití metod data augmentation (viz kapitola 4.5).

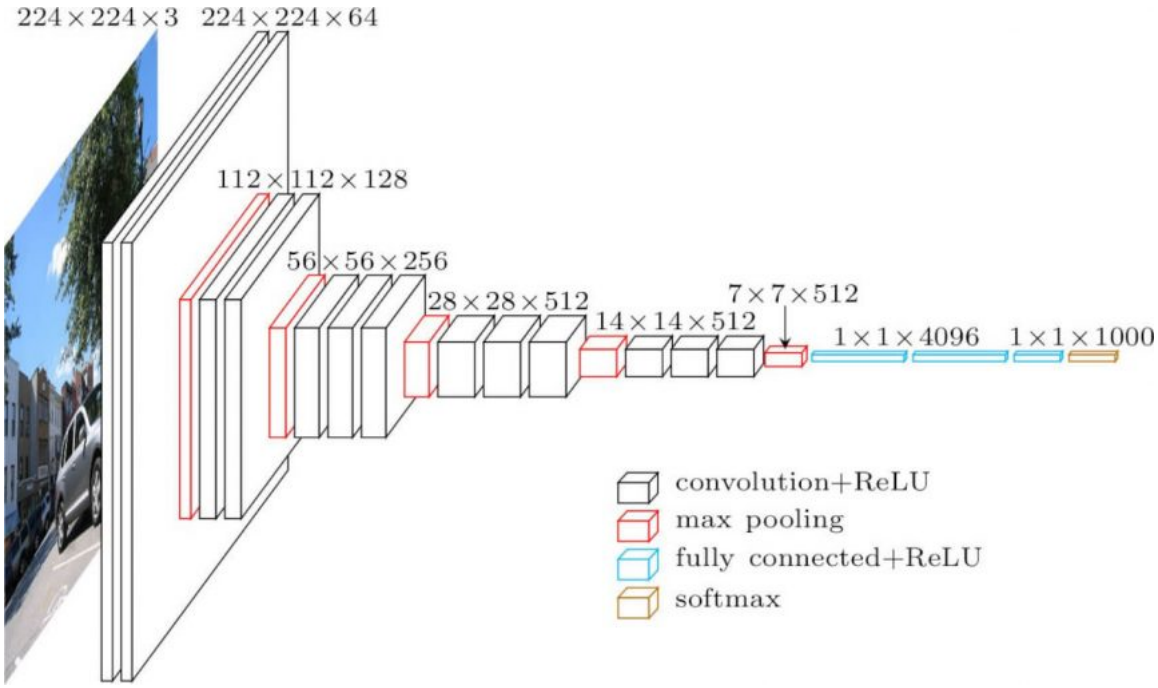
VGG16

Architektura konvoluční neuronové sítě VGG16 [33] je vytvořena pro klasifikaci fotografií o rozměru 224×224 px. Architektura VGG16 vychází z architektury AlexNet [18], ale místo použití konvolučních jader o rozměrech 11×11 a 5×5 používá více konvolučních vrstev s jádry o rozměrech 3×3. Na vstupní RGB obrázek o rozměrech $224 \times 224 \times 3$ je aplikováno celkem 13 konvolučních vrstev s aktivační funkcí ReLU a 5 max pooling vrstev. Po poslední max pooling vrstvě následují dvě plně propojené vrstvy po 4 096 perceptronech s aktivační funkcí ReLU a výstupní plně propojená vrstva se 133 perceptrony a aktivační funkcí softmax. Schéma VGG 16 architektury je na obrázku 4.3. Tato síť měla Top-1 úspěšnost 70,4 %.

ResNet

Architektura ResNet [19] je navržena pro klasifikaci fotografií o rozměru 224×224 px. Architektury jako AlexNet [18], VGG 16 nebo GoogLeNet [39] se nemohou neomezeně „prohlubovat“ (přidávat další vrstvy), protože při velkém počtu vrstev v sekvenčním modelu může nastat problém nekonečně malého gradientu [16] během algoritmu zpětné propagace (viz kapitola 3.6), což činí trénování takové neuronové sítě skoro nemožným.

Resnet se tomuto problému vyhýbá tím, že přidává zkratky (*identity shortcut connections*), přes které se data mohou vyhnout daným vrstvám. Tato data jsou pak přičtena k výstupu daných vrtev. Díky tomu může mít ResNet desítky vrstev, nejvíce se jich používá v architektuře ResNet 152, kde je 152 skrytých vrstev [32]. Schéma *Residual* bloku je vidět na obrázku 4.4. Tato síť měla Top-1 úspěšnost 77,1 %.



Obrázek 4.3: Schéma architektury VGG16. Převzato z neurohive.io [6].

GoogLeNet

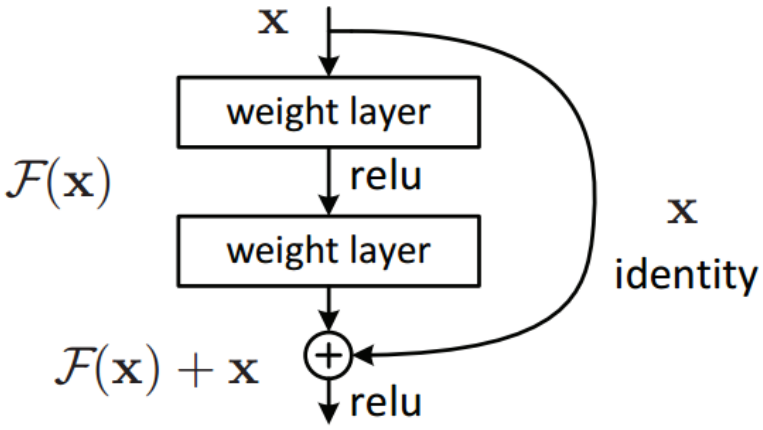
GoogLeNet [41] (někdy také jako Inception v1) je architektura, která ovládla The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014. Základem GoogLeNet architektury je *Incpetion blok*, který je znázorněn na obrázku 4.5. V dříve zmíněných architekturách byly rozměry konvolučního filtru pevně dané. V Inception bloku je na stejné úrovni použito několik různých velikostí konvolučních filtrů spolu s 3×3 max pooling vrstvou. Výsledky těchto operací se konkatenují ve výstupní vrstvě Inception bloku. Konkrétně v GoogLeNet architektuře se používají konvoluční filtry o velikostech 1×1, 3×3 a 5×5. Toto umožní neuronové síti hledat různě velké příznaky v každé vrstvě a až během trénování se rozhodnout, které příznaky (a jak velké) hledá. V každé vrstvě si neuronová síť podle hodnot vah daného příznaku rozhodne, zda jsou důležité rozměrově větší nebo menší příznaky.

Výše popsaný Inception blok má velký počet parametrů a je velmi náročný na výpočty. Proto GoogLeNet používá konvoluční vrstvy s rozměrem filtru 1×1. Tyto vrstvy slouží ke snížení počtu dimenzí, což snižuje počet parametrů sítě (zrychluje trénování) a zabraňuje přetrénování. Tyto 1×1 konvoluční vrstvy jsou vloženy před 3×3 a 5×5 konvoluční vrstvy a po 3×3 max pooling vrstvě. Tyto 1×1 konvoluční vrstvy pro snížení počtu dimenzí představují žluté bloky na obrázku 4.5.

Výsledná síť GoogLeNet obsahuje celkem 22 vrstev, což je více, než u VGG16 architektury, ale méně než u ResNet architektury. Celkově se skládá z několika Inception bloků, které jsou poskládané sekvenčně za sebou. Síť s touto architekturou měla Top-1 úspěšnost 78,6 %.

Inception-Resnet-V2

Koncept Residual bloku z architektury ResNet i koncept Inception bloku z GoogLeNet architektury jsou přelomové a v době svého představení vědecké komunitě vykazovaly tyto



Obrázek 4.4: Schéma *Residual* bloku z architektury ResNet. Převzato z towardsdatascience.com [31].

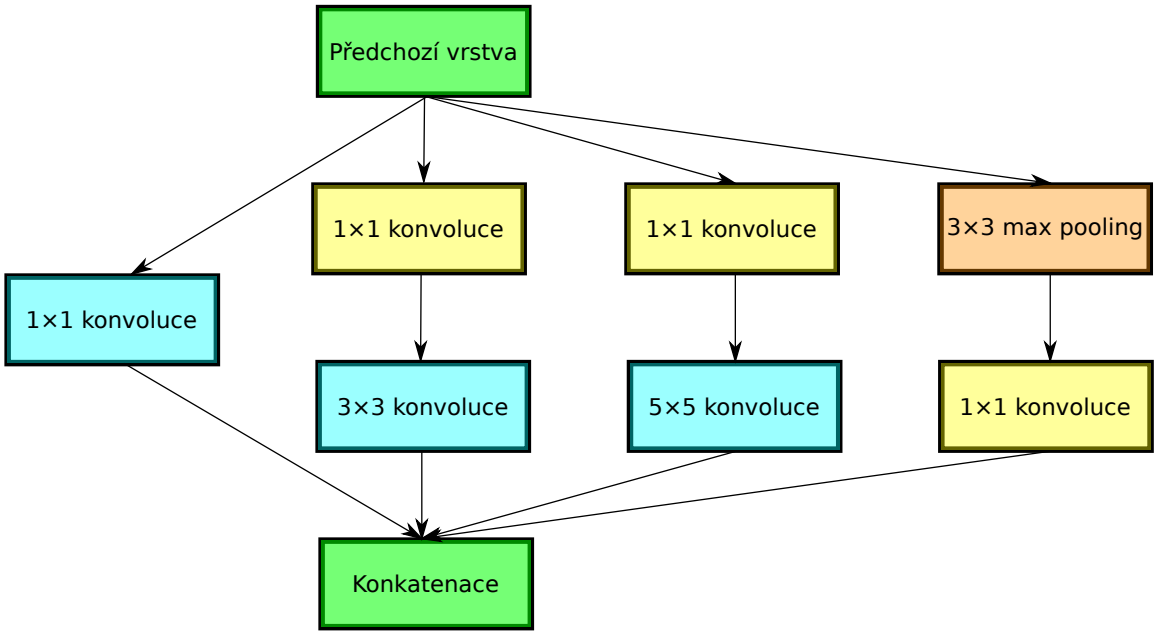
architektury nejlepší úspěšnost v soutěži The ImageNet Large Scale Visual Recognition Challenge. Architektura Inception-Resnet-V2 [38] přebírá jak koncept Residual bloku z architektury ResNet, tak koncept Inception bloku z architektury GoogLeNet. Schéma této architektury je na obrázku 4.6. Tato architektura se ukázala jako nejpřesnější v Top 1 přesnosti i Top 5 přesnosti. Na datové sadě ImageNet dosahuje Top 1 úspěšnost 80,4 %. Pro úlohu klasifikace modelu automobilu se mi povedlo tuto architekturu natrénovat s validační úspěšností 80,7 %.

4.5 Data augmentation

Neuronové sítě potřebují často poměrně hodně trénovacích dat, abychom zamezili přetrénování klasifikátoru. Přetrénování klasifikátoru je pojem, který znamená, že daný klasifikátor podává dobré výsledky na trénovacích datech, ale znatelně horší výsledky na validačních (neviděných datech). Ukázku trénovacích křivek najdete na obrázku 4.7. Vlevo vidíme proces trénování, při kterém dochází k přetrénování. To poznáme tak, že se chybovost na validačních datech začne zvyšovat, zatímco chybovost na trénovacích datech stále klesá. Graf napravo na rozdíl ukazuje proces trénování, kdy nedochází ke znatelnému přetrénování klasifikátoru.

Data augmentation metody jsou mocným nástrojem, který může pomoct zabránit přetrénování neuronové sítě. Nejedná se o určitou konkrétní techniku nebo algoritmus. Principem data augmentation je umělé zvětšování původní trénovací datové sady. Jedná se například o techniky jako obrácení původní fotografie, rotaci, přiblížení, úpravu barev, přidání šumu, vynechání části fotografie či kombinování různých trénovacích fotografií... V následujícím seznamu uvádím techniky data augmentation, které byly použity při trénování výsledného klasifikátoru:

- **Horizontální převrácení**
- **Rotace**
- **Přiblížení**



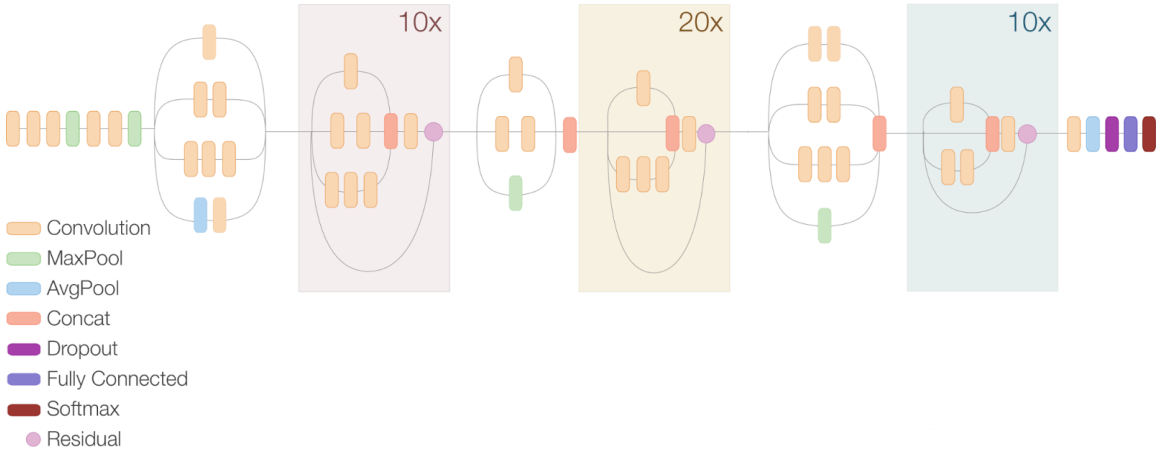
Obrázek 4.5: Schéma Inception bloku v GoogLeNet architektuře.

- **Úprava saturace**
- **Úprava jasu**
- **Vypuštění části obrazu** – technika zvaná *Image drop* navržená v článku [35]
- **Přidání náhodného šumu**

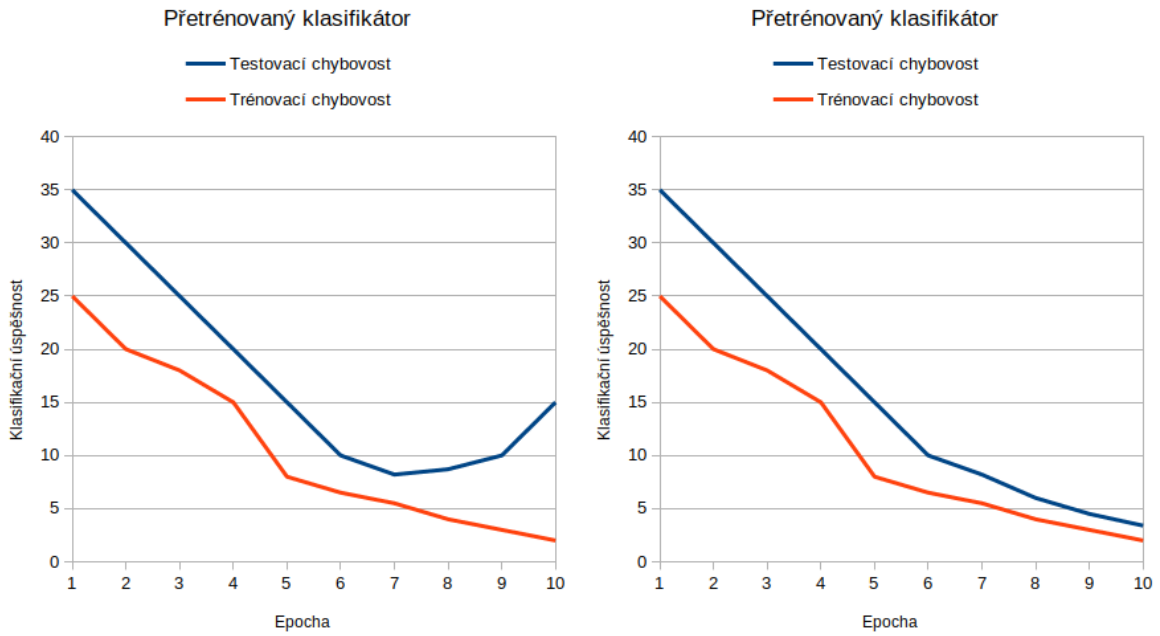
Na obrázku 4.8 je vidět ukázka fungování metod data augmentation vypsaných v seznamu výše. Obrázek vlevo nahoře je původní, ostatní jsou vytvořeny aplikací náhodných úprav ze seznamu výše.

4.6 Shrnutí výsledného řešení

Výsledné řešení je implementováno v jazyce Python a používá knihovnu Tensorflow. Trénovací datová sada byla vybrána datová sada *BoxCars116k* [35]. Trénování proběhlo na výpočetních prostředcích Metacentra. Klasifikátor je implementován jako konvoluční neuronová síť. Vstupní obrázky mají velikost 299×299 px. Je použita architektura Inception-Resnet-V2, za kterou následuje plně propojená vrstva se 133 perceptrony. Za touto plně propojenou vrstvou následuje aktivační funkce softmax. Při trénování byly použity techniky data augmentation jako rotace, přiblížení, vypuštění části obrazu a další... Výsledný klasifikátor rozlišuje mezi 133 různými modely automobilů. Na validačních (neviděných) datech má úspěšnost 80,7 %.



Obrázek 4.6: Schéma Inception-Resnet-V2 architektury. Převzato z ai.googleblog.com [4].



Obrázek 4.7: Učící křivky s přetrénováním (nalevo) a učící křivky s požadovaným vztahem trénovací a testovací křivky (napravo)

Obrázek 4.8: Ukázka fungování vybraných data augmentation metod. Obrázek vlevo nahoře je originální, ostatní jsou z něj vytvořeny náhodnými úpravami.

Kapitola 5

Vyhodnocení úspěšnosti

Tato kapitola vznikla proto, aby ukazovala, že výsledný model (popsaný v kapitole 4) je použitelný pro klasifikaci modelu automobilu na reálných datech. V kombinaci s vhodným detektorem automobilu je toto řešení aplikovatelné na záznam reálného dopravního provozu pořízený na běžný mobilní telefon (se stativem).

Pro vyhodnocení úspěšnosti vytvořeného klasifikátoru byla vytvořena vlastní dílčí datová sada. Tato datová sada obsahuje 1 034 fotografií celkem ze 36 tříd. Klasifikátor dosáhl úspěšnosti 93,5 % (Top-5).

5.1 Pořízení vlastní datové sady

Datová sada vznikla nahráváním dopravního provozu v Brně. Nahrané video má rozlišení 1920 x 1080 px (30 FPS). Videozáznam byl nahrán na dálničním mostě, provoz je zachycen z několika různých úhlů (včetně předního, zadního i bočního).

Z nahraného video záznamu jsem vybíral jednotlivé snímky (s frekvencí 6 FPS). Na tyto snímky jsem aplikoval detektor automobilů (viz kapitola 5.2). Tyto snímky automobilů jsem ručně třídil mezi třídy odpovídající původní datové sadě (viz kapitola 4.1). Ukázka fotografií z této datové sady je na obrázku 5.1.

5.2 Detekce automobilu

Na snímky (které obsahují vozovku i její okolí) byl aplikován detektor automobilů. Tento detektor je implementován pomocí OpenCV (viz kapitola 4.2.4). Vybrané příznaky pro automobily jsou předem natrénované a byly převzaty z repozitáře od Abhishek Kumar Annamraju[1]. Jedná se o tzv. Haarovy příznaky [42], které fungují na principu konvoluce.

5.3 Vyhodnocení úspěšnosti

Vyhodnocení úspěšnosti klasifikátoru, jehož vznik byl popsán v kapitole 4, proběhlo porovnáním odhadovaných tříd a skutečných tříd. Výsledná Top-5 přesnost je 93,5 %, což se výrazně neliší od validačních dat získaných během trénování (viz kapitola 4.6). Používám zde Top-5 metriku, protože datovou sadu jsem vytvořil pravděpodobně ne zcela přesně. Je možné, že označení některých automobilů není správné, ale označuje model automobilu,

[1] https://gist.github.com/199995/37e1e0af2bf8965e8058a9dfa3285bc6

Obrázek 5.1: Série náhodně vybraných fotografií z vytvořené datové sady

který je vizuálně podobný (např. modely Hyundai i20 a Hyundai i30). Metrika Top-5 tuto chybu v označení pravděpodobně nebude brát jako chybu klasifikace (jsou-li si dané modely podobné).

Kapitola 6

Experimenty

6.1 Úspěšnost jednotlivých modelů automobilů

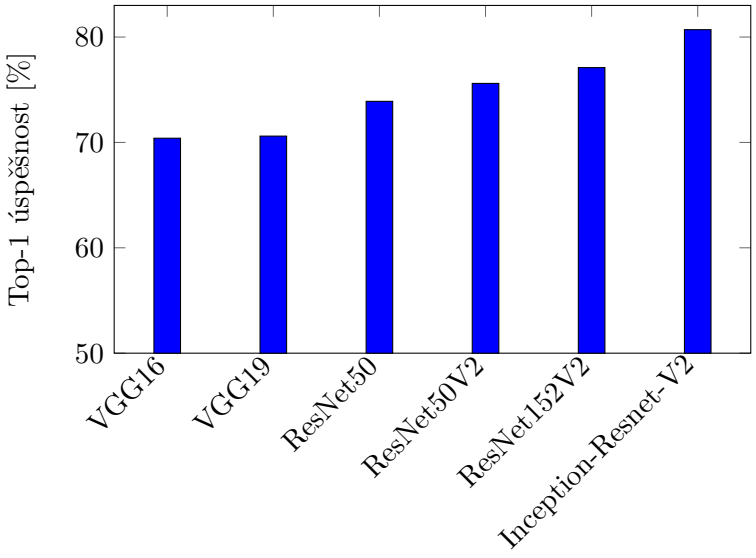
Tabulka 6.1 ukazuje Top-5 úspěšnost výsledného modelu na nově vytvořené datové sadě. Z ní lze vypozorovat jisté tendence. Přesnost na některých modelech je vyšší, než na jiných. Např. modely výrobce Škoda mají všechny úspěšnost přes 90 % a počet vozidel Škoda zachycených v datové sadě je vyšší, než u ostatních výrobců.

Audi	100 %
BMW	0 %
Citroën	97 %
Fiat	52,6 %
Ford	98,6 %
Honda	77,7 %
Hyundai	89,4 %
Kia	90.9 %
Mercedes-benz	100 %
Opel	92 %
Peugeot 206	94,7 %
Renault	91,5 %
Škoda	95,9 %
Toyota	100 %
Volkswagen	79,7 %
Volvo	100 %

Obrázek 6.1: Úspěšnost klasifikátoru na jednotlivých třídách

6.2 Úspěšnost na převzatých architekturách

Na obrázku 6.2 je graf Top-1 úspěšnosti na vybraných převzatých architekturách. Jedná se o trénování na stejných datech, jako v kapitole 4.5. Na grafu je zřetelný rozdíl v přesnosti mezi VGG a ResNet architekturou a dále mezi ResNet a Inception-Resnet-V2 architekturou. Použitá architektura Inception-Resnet-V2 dosahuje nejlepších výsledků.



Obrázek 6.2: Top-1 úspěšnost převzatých architektur

Kapitola 7

Závěr

Tato práce obsahuje popis existujících přístupů k problému klasifikace modelu automobilu. Z nich se zaměřuje na klasifikaci pomocí konvolučních neuronových sítí. Práce také obsahuje popis jednotlivých částí, ze kterých se taková konvoluční neuronová síť skládá a je zde popsán proces, během kterého došlo k vytvoření a natrénování vhodného klasifikátoru.

V této práci bylo navrhnuto a implementováno řešení, které dokáže na videozáznamu najít automobily a poznat jejich výrobce a model s přesností 80,7 %, což je podobná úspěšnost, ke které dochází autoři ostatních klasifikátorů. [34] [36]

K této práci je mimo jiné vytvořena dílčí datová sada, která souží k ověření, že vytvořený model je schopen fungovat i na jiné, než na trénovací datově sadě. Datová sada obsahuje 1 034 fotografií ze 36 různých tříd. Výsledný model na této dílčí datové sadě dosáhl Top-5 úspěšnosti 93,5 %. Výsledkem je program, který dokáže na videozáznamu nalézt automobily a přiřadit k nim jméno výrobce a modelu automobilu.

Bylo provedeno několik experimentů, které popisují, jak se mění úspěšnost klasifikátoru za různých okolností. Nakonec byl vytvořen plakát a ukázkové video, které předvádí funkčnost systému.

Navázání na práci popsanou v této zprávě je možné například rozšířením trénovací datové sady o nové modely automobilů. Pro zlepšení přesnosti bych se zaměřil na normalizaci dat před vstupem do klasifikátoru. Metoda *Unpack* (viz kapitola 2.1) přináší razantní zlepšení při použití jednodušších architektur (jako AlexNet, VGG16 a další...), ovšem při použití složitějších architektur (jako ResNet) její přínos slábne. Pomoct by mohlo také využívání informací z videozáznamu, pokud bychom sledovali trajektorii vozidla během celého průjezdu automobilu. Ke klasifikaci bychom mohli využít vícero záběrů z různých úhlů (jak automobil projíždí).

Literatura

[1] *Image classification : TensorFlow Core*. Dostupné z: https://www.tensorflow.org/tutorials/images/classification.

[2] *Max-pooling / Pooling*. Dostupné z: https://computerscience wiki.org/index.php/Max-pooling_/_Pooling.

[3] *Tf.keras.preprocessing.image.ImageDataGenerator*. Dostupné z: https://tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.

[4] *Improving Inception and Image Classification in TensorFlow*. Aug 2016. Dostupné z: https://ai.googleblog.com/2016/08/improving-inception-and-image.html.

[5] *Deep Learning in Five and a Half Minutes*. Jul 2018. Dostupné z: https://www.edge-ai-vision.com/2018/07/deep-learning-in-five-and-a-half-minutes/.

[6] *VGG16 - Convolutional Network for Classification and Detection*. Nov 2018. Dostupné z: https://neurohive.io/en/popular-networks/vgg16/.

[7] Budhiraja, A. Learning Less to Learn Better-Dropout in (Deep) Machine learning. *Medium*. Medium. Mar 2018. Dostupné z: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[8] Clady, X., Negri, P., Milgram, M. a Poulenard, R. Multi-class Vehicle Type Recognition System. In: Prevost, L., Marinai, S. a Schwenker, F., ed. *Artificial Neural Networks in Pattern Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 228–239.

[9] Dalal, N. a Triggs, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 2005, sv. 1, s. 886–893 vol. 1.

[10] Dubská, M., Sochor, J. a Herout, A. Automatic Camera Calibration for Traffic Understanding. In: *Proceedings of BMVC 2014*. 2014, s. 1–10.

[11] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R. a Lin, C.-J. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*. Srpen 2008, sv. 9, s. 1871–1874.

[12] Fang, J., Zhou, Y., Yu, Y. a Du, S. Fine-Grained Vehicle Model Recognition Using A Coarse-to-Fine Convolutional Neural Network Architecture. *IEEE Transactions on Intelligent Transportation Systems*. 2017, sv. 18, č. 7, s. 1782–1792.

[13] Felzenszwalb, P. F., Girshick, R. B., McAllester, D. a Ramanan, D. Object Detection with Discriminatively Trained Part-Based Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2010, sv. 32, č. 9, s. 1627–1645.

[14] Fuat. Google Colab Free GPU Tutorial. *Medium*. Deep Learning Turkey. Mar 2019.

[15] Fukushima, K. a Miyake, S. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*. 1982, sv. 15, č. 6, s. 455 – 469. DOI: https://doi.org/10.1016/0031-3203(82)90024-3. ISSN 0031-3203. Dostupné z: http://www.sciencedirect.com/science/article/pii/0031320382900243.

[16] Fung, V. An Overview of ResNet and its Variants. *Medium*. Towards Data Science. Jul 2017.

[17] Geirhos, R., Janssen, D. H. J., Schütt, H. H., Rauber, J., Bethge, M. et al. Comparing deep neural networks against humans: object recognition when the signal gets weaker. *CoRR*. 2017, abs/1706.06969.

[18] Hao, G. A Walk-through of AlexNet. *Medium*. Medium. Aug 2017.

[19] He, K., Zhang, X., Ren, S. a Sun, J. Deep Residual Learning for Image Recognition. *CoRR*. 2015, abs/1512.03385.

[20] Krause, J., Stark, M., Deng, J. a Fei Fei, L. 3D Object Representations for Fine-Grained Categorization. In: *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*. Sydney, Australia: [b.n.], 2013.

[21] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. a Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, s. 1097–1105.

[22] Liao, L., Hu, R., Xiao, J., Wang, Q., Xiao, J. et al. Exploiting effects of parts in fine-grained categorization of vehicles. In: *2015 IEEE International Conference on Image Processing (ICIP)*. 2015, s. 745–749.

[23] Lin, M., Chen, Q. a Yan, S. Network In Network. Prosinec 2013.

[24] Lin, Y.-L., Morariu, V. I., Hsu, W. a Davis, L. S. Jointly Optimizing 3D Model Fitting and Fine-Grained Classification. In: *Computer Vision – ECCV 2014*. Springer International Publishing, 2014, s. 466–480.

[25] Liu, D. A Practical Guide to ReLU. *Medium*. Medium. Nov 2017.

[26] Llorca, D. F., Colás, D., Daza, I. G., Parra, I. a Sotelo, M. A. Vehicle model recognition using geometry and appearance of car emblems from rear view images. In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 2014, s. 3094–3099.

[27] O'Shea, K. a Nash, R. An Introduction to Convolutional Neural Networks. *ArXiv e-prints*. Listopad 2015.

[28] Perronnin, F., Sánchez, J. a Mensink, T. Improving the Fisher Kernel for Large-Scale Image Classification. In:. Září 2010, sv. 6314, s. 143–156.

[29] Ruder, S. An overview of gradient descent optimization algorithms. *CoRR*. 2016, abs/1609.04747. Dostupné z: http://arxiv.org/abs/1609.04747.

[30] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S. et al. ImageNet Large Scale Visual Recognition Challenge. *CoRR*. 2014, abs/1409.0575.

[31] Sahoo, S. *Residual blocks-Building blocks of ResNet*. Towards Data Science, Nov 2018. Dostupné z: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec.

[32] Shorten, C. Introduction to ResNets. *Medium*. Towards Data Science. May 2019.

[33] Simonyan, K. a Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2014. cite arxiv:1409.1556.

[34] Sochor, J., Herout, A. a Havel, J. BoxCars: 3D Boxes as CNN Input for Improved Fine-Grained Vehicle Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 3006–3015.

[35] Sochor, J., Špaňhel, J. a Herout, A. BoxCars: Improving Fine-Grained Recognition of Vehicles Using 3-D Bounding Boxes in Traffic Surveillance. *IEEE Transactions on Intelligent Transportation Systems*. 2018, PP, č. 99, s. 1–12. DOI: 10.1109/TITS.2018.2799228. ISSN 1524-9050.

[36] Sochor, J., Špaňhel, J. a Herout, A. BoxCars: Improving Fine-Grained Recognition of Vehicles Using 3-D Bounding Boxes in Traffic Surveillance. *IEEE Transactions on Intelligent Transportation Systems*. 2019, sv. 20, č. 1, s. 97–108.

[37] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. a Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014, sv. 15, č. 56, s. 1929–1958.

[38] Szegedy, C., Ioffe, S., Vanhoucke, V. a Alemi, A. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016.

[39] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E. et al. Going Deeper with Convolutions. *CoRR*. 2014, abs/1409.4842.

[40] TORRES.AI, J. Learning process of a neural network. *Medium*. Towards Data Science. Dec 2018.

[41] Tsang, S.-H. *Review: GoogLeNet (Inception v1)- Winner of ILSVRC 2014 (Image Classification)*. Coinmonks, Jun 2019.

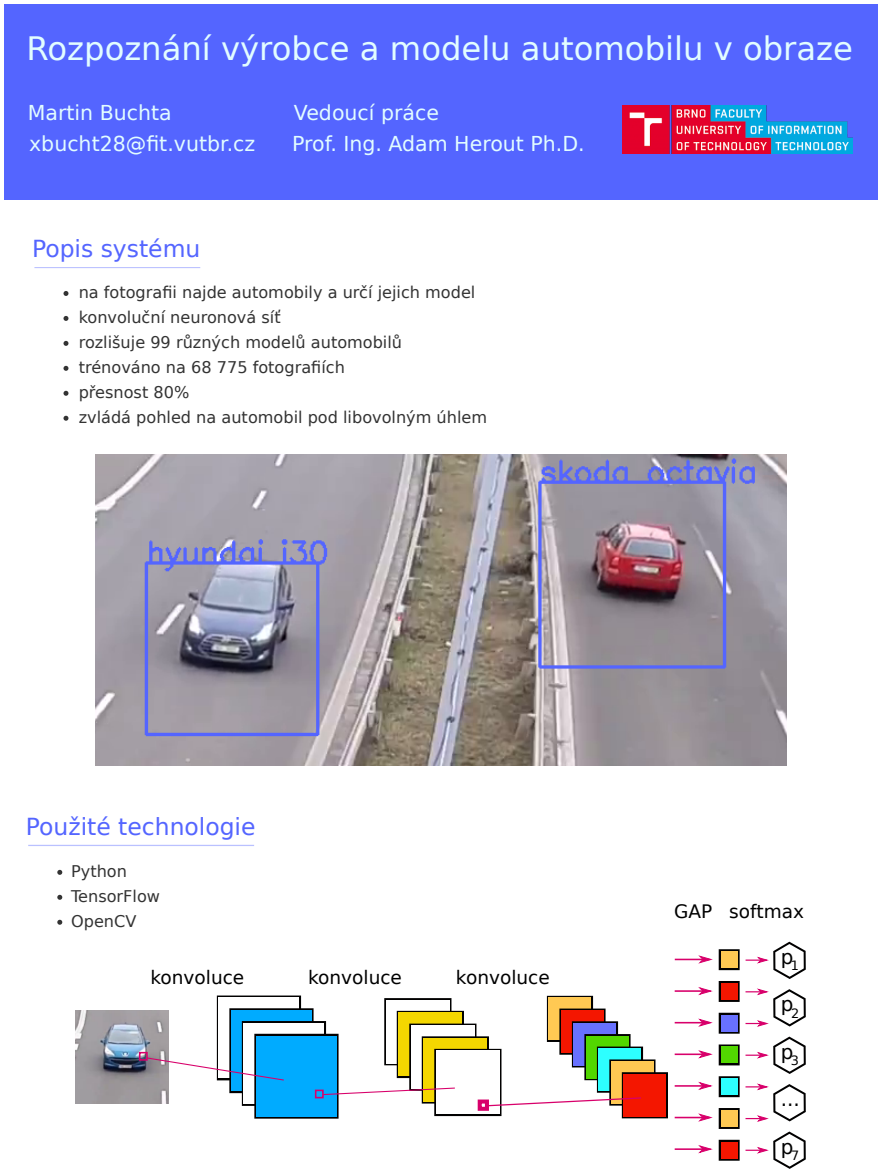
[42] Wang, C.-F. *What's the Difference Between Haar-Feature Classifiers and Convolutional Neural Networks?* Towards Data Science, Aug 2018. Dostupné z: https://towardsdatascience.com/whats-the-difference-between-haar-feature-classifiers-and-convolutional-neural-networks-ce6828343aeb.

[43] Wu, J. Introduction to Convolutional Neural Networks. In:. 2017.

[44] Yang, L., Luo, P., Loy, C. C. a Tang, X. A Large-Scale Car Dataset for Fine-Grained Categorization and Verification. *CoRR*. 2015, abs/1506.08959. Dostupné z: http://arxiv.org/abs/1506.08959.

Příloha A

Plakát



Obrázek A.1: Plakát

Příloha B

Obsah paměťového média

- *code/* – Zdrojové kódy obsahující stahování, rozbalování a přípravu datové sady, vytváření a trénování modelu, vyhodnocení modelu a zpracování videa, které ve videozáznamu najde automobily, označí je čtverečkem a popiskem modelu automobilu.
- *dataset/* – Vytvořená datová sada pro ověření funkčnosti klasifikátoru.
- *latex/* – Zdrojové soubory pro vytvoření textové zprávy.
- *plakat/* – Plakát shrnující tuto práci.
- *saved_model/* – Natrénovaný model.
- *video/* – Video shrnující tuto práci a výstup programu na reálném záběru silnice.