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

KLASIFIKACE FOTOGRAFIÍ POMOCÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ

DEEP LEARNING FOR IMAGE CLASSIFICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JANA ZIKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL HRADIŠ, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Ziková Jana**
Obor: Informační technologie
Téma: **Klasifikace fotografií pomocí hlubokých neuronových sítí**
Deep Learning for Image Classification
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte základy neuronových sítí a back-propagation.
2. Vytvořte si přehled o současných metodách pro tvorbu hlubokých sítí, a hlavně konvolučních sítí.
3. Vyberte konkrétní metodu aplikovatelnou na klasifikaci fotografií.
4. Obstarejte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručný plakát prezentující vaši práci, její cíle a výsledky.

Literatura:

- Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- http://www.image-net.org/challenges/LSVRC/2013/results.php

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hradiš Michal, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá klasifikací produktů internetového obchodu s pomocí jejich fotografií. K tomuto účelu využíváme existující modely hlubokých konvolučních neuronových sítí. Cílem práce bylo navrhnout experimenty, které povedou k co největší úspěšnosti při klasifikaci fotografií produktů.

Abstract

This bachelor thesis deals with electronic commerce website products classification using product's photographs. For this purpose we use already implemented models of deep convolutional neural networks. Tho goal of this theses is to design experiments that will lead to the best possible results in product images classification.

Klíčová slova

Hluboké neuronové sítě, konvoluční neuronové sítě, klasifikace obrazu, Inception v3, Inception-ResNet v2, MobileNet, CDiscount.

Keywords

Deep neural networks, convolutional neural networks, image classification, Inception v3, Inception-ResNet v2, MobileNet, CDiscount.

Citace

ZIKOVÁ, Jana. *Klasifikace fotografií pomocí hlubokých neuronových sítí.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Klasifikace fotografií pomocí hlubokých neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Michala Hradiše, Ph.D. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Jana Ziková
17. května 2018

Poděkování

Tímto bych chtěla poděkovat svému vedoucímu práce Ing. Michalovi Hradišovi, Ph.D. za jeho pomoc, vedení, rady a věnovaný čas bez kterých by vznik této práce nebyl možný. Dále bych chtěla poděkovat výpočetnímu centru MetaCentrum: Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Obsah

Kapitola 1

Úvod

V poslední době dochází k velkému rozvoji v odvětví počítačového vidění, které se využívá v mnoha oblastech. Jednou z podúloh počítačového vidění je i klasifikace obrazu. Příkladem využití klasifikace obrazu je identifikace objektů na fotografiích, kterou se budeme v této práci zabývat.

Nejefektivnější existující metodou zpracování obrazu jsou dnes konvoluční neuronové sítě. Existuje množství dostupných implementovaných, a dokonce i natrénovaných, modelů konvolučních neuronových sítí, které se dají stáhnout a použít k vytvoření vlastního klasifikátoru. Asi všechny z nich však mají společné to, že byly natrénovány na datových sadách vytvořených speciálně pro ten účel.

V této práci budeme mít k dispozici reálná data poskytnutá z databáze internetového obchodu, která neprošla žádnou zvláštní úpravou či kontrolou. Budeme tedy moci vyzkoušet, jak moc jsou existující modely konvolučních sítí aplikovatelné na reálná data a pomocí experimentů se budeme snažit zjistit, jak je potřeba upravit postup trénování sítí, aby fungovalo i na datech, která k tomu nebyla přímo určena.

Nejdříve se podíváme na data, se kterými budeme pracovat a zkusíme o nich co nejvíce zjistit. Potom si řekneme něco málo o konvolučních neuronových sítích a rozebereme si konkrétní modely, které budou v této práci použity. Nakonec na základě znalostí získaných analýzou datové sady navrhneme a provedeme experimenty týkající se trénování našich modelů a zhodnotíme jejich výsledky.

Kapitola 2

Cdiscount's Image Classification Challenge

Cdiscount's Image Classification Challenge je soutěž zadaná francouzskou společností *Cdiscount*. Společnost se věnuje elektronickému obchodu a má velmi široký záběr produktů. Odhadem momentálně nabízí přes 30 milionů produktů z různých kategorií. Z důvodu rozšiřovaní nabídky a snahy nalézt nový způsob efektivní kategorizace produktů vypsala v září 2017 soutěž[1], do které poskytla část svého katalogu produktů. Data nebyla pro účely soutěže nijak upravována ani speciálně kontrolována. Jedná se o reálná data převzatá přímo z databáze společnosti a prakticky využívaná v jejím internetovém obchodě.

Data poskytnutá do soutěže obsahují téměř 9 milionů produktů z více než 5 000 kategorií. Přitom ke každému produktu přísluší až čtyři fotografie. Celkově tak dostaneme přes 15 milionů obrázků. Cílem soutěže bylo vytvořit model, který bude schopný co nejlépe automaticky klasifikovat produkty do příslušných kategorií, a to pouze v závislosti na fotografiích daných produktů. Měřítkem úspěšnosti je zde přitom podíl správně zařazených produktů vůči celku.

Poskytnutá data jsou rozdělená na dvě části – trénovací a testovací. Trénovací část obsahuje ke každému produktu identifikátor kategorie, do které patří, v testovací části tato informace není. Jediný způsob, jak zjistit, zda byla testovací data klasifikována správně, je nechat si je vyhodnotit v rámci soutěže. To v mé práci nebylo možné, neboť jsem se soutěže přímo neúčastnila, a navíc soutěž skončila již v prosinci 2017. Pracuji tedy pouze s trénovací částí dat. Ta představuje zhruba 80 % všech dat, těsně přes 7 milionů produktů s 12 miliony fotografiemi. Testovací část už dále nebude brána v úvahu.

2.1 Analýza datasetu

Dataset používaný v této práci sestává z 12 371 293 obrázků patřících k 7 069 896 produktům. Data jsou rozdělena podle produktů, ke kterým náleží, tak, že každý produkt má své identifikační číslo, číslo kategorie, do které náleží, a 1 až 4 obrázky v rozlišení 180x180 pixelů. V rozbalené podobě to představuje 378 GB dat. Na obrázku 2.1 je vidět prvních 20 obrázků z datasetu.

Rozdělení produktů podle počtu obrázků. Většina produktů, konkrétně 4 369 441 produktů, obsahuje pouze jeden obrázek, další téměř třetina má buď dva (1 128 588 pro-

[1] https://www.kaggle.com/c/cdiscount-image-classification-challenge

Obrázek 2.1: Ukázka obrázků z datasetu. Popisky obrázků jsou ve francouzštině.

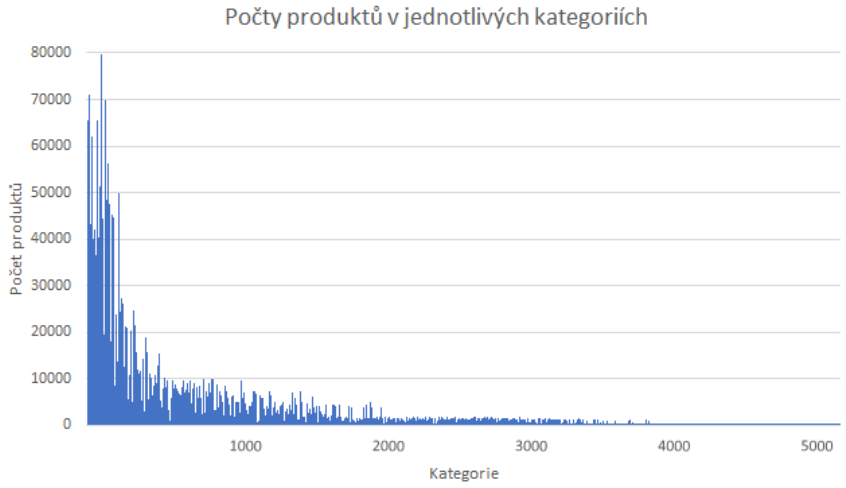
id kategorie	1. úroveň	2. úroveň	3. úroveň
1000021794	ABONNEMENT / SERVICES	CARTE PREPAYEE	CARTE PREPAYEE MULTIMEDIA
1000012764	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	ABRI FUMEUR
1000012776	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	ABRI VELO - ABRI MOTO
1000012768	AMENAGEMENT URBAIN - VOIRIE	AMENAGEMENT URBAIN	FONTAINE A EAU
1000012755	AMENAGEMENT URBAIN - VOIRIE	SIGNALETIQUE	PANNEAU D'INFORMATION EXTERIEUR

Tabulka 2.1: Ukázka hierarchie kategorií. Názvy kategorií jsou ve francouzštině.

duktů) nebo čtyři (1 029 075 produktů) obrázky, a nejméně (541 792 produktů) je produktů se třemi obrázky. Histogram rozdělení můžeme vidět na obrázku 2.2.

Kategorie Produkty jsou rozděleny celkem do 5 270 kategorií. Kategorie mají hierarchické rozdělení na tři úrovně. Identifikační čísla kategorií, které jsou přiřazené ke všem produktům, patří nejnižší úrovni hierarchie. Další úrovně pro nás tedy v této práci nejsou nijak zvlášť užitečné, mohly by však být využitelné při případné další práci s daty. Ukázka hierarchie kategorií je v tabulce 2.1.

Obrázek 2.2: Histogram rozdělení produktů podle počtu obrázků, které obsahují.

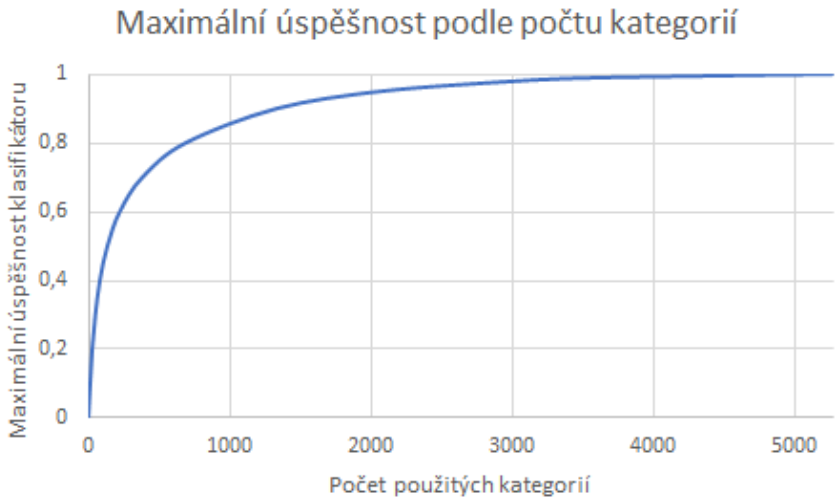


Obrázek 2.3: Histogram rozdělení produktů do jednotlivých kategorií.

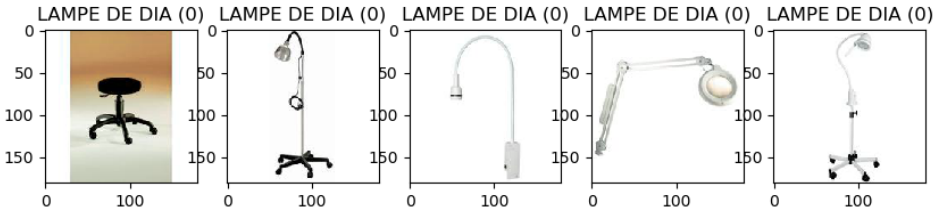
Rozdělení produktů do kategorií Jak je vidět z grafu na obrázku 2.3, zastoupení jednotlivých kategorií je velmi nerovnoměrné. Největší kategorie obsahuje 79 640 produktů a naopak máme 31 kategorií, které každá obsahují jen 12 produktů.

Z celkových 5 270 kategorií je pouze 1 034 takových, které obsahují více než 1 000 produktů, přičemž 143 z nich obsahuje více než 10 000 produktů. 100 největších kategorií dává dohromady 3 134 704 produktů a 134 největších kategorií dokonce tvoří více než polovinu všech produktů (celkem 3 540 572 produktů). Naopak je zde 1 780 kategorií, které mají méně než 100 produktů a dokonce 258 kategorií s méně než 20 produkty. 100 nejmenších kategorií dává dohromady pouze 1 300 produktů.

Chyby v datasetu Jak již bylo zmíněno, data pochází z databáze společnosti Cdiscount a před poskytnutím do soutěže nebyla nijak upravená. Tato data byla automaticky klasifikována pomocí blíže nespecifikovaných algoritmů využívajících strojové učení, které pracovaly



Obrázek 2.4: Graf závislosti maximální dosažitelné kategorie na počtu použitých kategorií. Předpokládáme, že kategorie jsou řazené od největší po nejmenší.



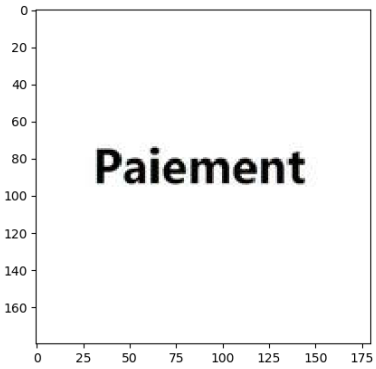
Obrázek 2.5: Ukázka kategorie obsahující chybně zařazený produkt. Popisky obrázků jsou ve francouzštině.

Obrázek 2.6: Ukázka kategorie obsahující chybně zařazený produkt. Popisky obrázků jsou ve francouzštině.

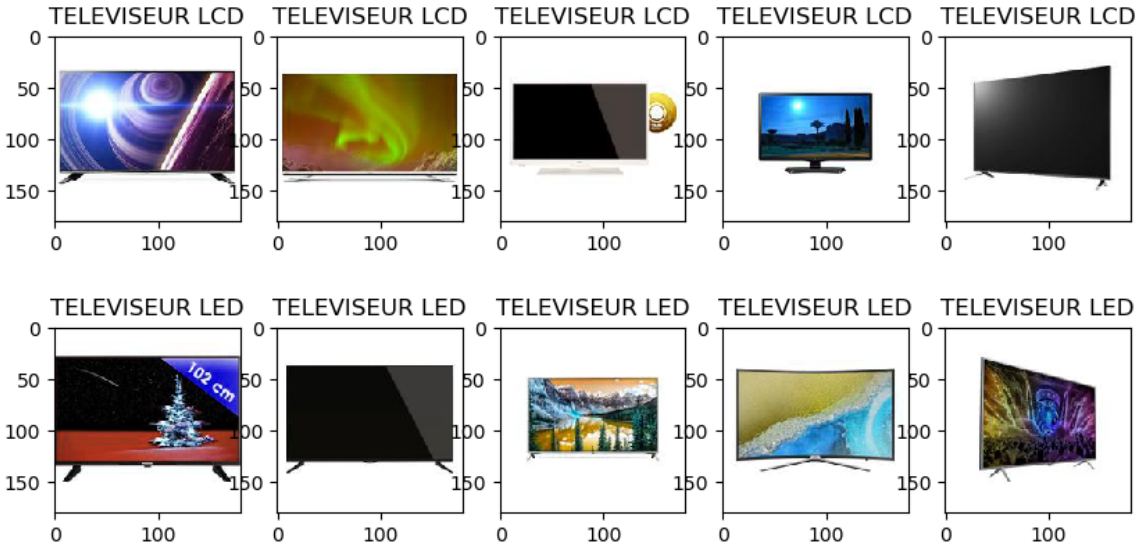
s textovými popisy produktů. Už jen to, že společnost hledá jiný způsob automatické klasifikace napovídá, že ten dosavadní nefunguje perfektně. Produkty jsou zařazeny do kategorie, pokud klasifikátor dokáže vybrat kategorii s alespoň 90 % pravděpodobností, přičemž této pravděpodobnosti dosáhne zhruba ve dvou třetinách případů. S ohledem na tuto skutečnost můžeme u dat očekávat určitou chybovost. Příklady produktů zařazených do nevhodných kategorií jsou na obrázcích 2.5, 2.6 a 2.7.

Problémem by také mohlo být, že produkty byly klasifikovány jen na základě textových popisů. To znamená, že nedocházelo k žádné kontrole obrázků u produktů, jejich kvalita

Obrázek 2.7: Ukázka kategorie obsahující chybně zařazený produkt. Popisky obrázků jsou ve francouzštině.



Obrázek 2.8: Ukázka výplňového obrázku. Tento obrázek se v datasetu vyskytuje na více místech a vždy je jediným obrázkem u příslušného produktu.



Obrázek 2.9: Ukázka dvou prakticky nerozlišititelných kategorií. Obrázky na stejném řádku vždy patří do stejné kategorie. Popisky obrázků jsou ve francouzštině.

a provedení se liší, pravděpodobně v závislosti na dodavateli. Mezi produkty dokonce najdeme takové, které obsahují pouze prázdný obrázek, viz. obrázek 2.8.

Rozlišitelnost kategorií Vzhledem k tomu, že jsme při klasifikaci omezení pouze na informace získané z obrázků, může být rozlišení některých kategorií velmi obtížné, pokud je vůbec možné. Ukázku jednoho takového případu máme na obrázku 2.9, kde vidíme dva typy (dvě různé kategorie) televizí. Je očividné, že rozlišení těchto kategorií pouze na základě obrázků, není dost dobře možné ani pro nás, nemůžeme to tedy očekávat od klasifikátoru. Dalším takovým případem může být například rozdělení CD s hudbou do kategorií podle žánru hudby.

Shrnutí Data, se kterými budeme pracovat, nejsou dokonalá. Hlavními problémy jsou nevyrovnanost v zastoupení jednotlivých kategorií, chybně zařazené produkty a příliš velká podobnost některých kategorií. Tyto problémy však většinou buď nejsou reálně řešitelné (například k rozlišení produktů z podobných kategorií bychom potřebovali další dodatečné informace, které nemáme) nebo se jejich řešení prostě nevyplatí (kontrola kategorií všech 7 milionů produktů a jejich případná oprava by byla příliš časově náročná). Nemůžeme tedy počítat se stoprocentní úspěšností klasifikátoru, odhadem můžeme reálně očekávat nejvýše 80 %.

2.2 Přístupy existujících řešení

Přestože se soutěže zúčastnilo poměrně velké množství lidí, vzhledem k tomu, že se jednalo o komerční nikoli akademickou záležitost, neexistují žádné práce popisující jejich postup a použité metody. Někteří z ešitelů však sdíleli alespoň nástin své strategie řešení v diskuzích u soutěže.

Obecně preferovanou metodou řešení je použití některého z před-trénovaných modelů hlubokých konvolučních sítí. Často jsou použity i soubory několika takovýchto modelů. K modelům jsou většinou přistavěny vlastní klasifikační vrstvy, ale jsou využity i čistě jen jako extraktory vlastností obrázků. Do struktury modelů samotných naprostá většina řešitelů nezasahuje. Při trénování sítí někteří z řešitelů používají konkatenaci jednotlivých obrázků u produktu do jednoho obrázku. Augmentace vstupních obrázků příliš využívána není, nejvýše jsou použity náhodné výřezy a překlopení obrázku. U testování je často využívané průměrování pravděpodobností jednotlivých kategorií obrázků patřících ke stejnému produktu.

Dosažená úspěšnost nejlepších modelů je mezi 79a 80 %. Počet epoch nad trénovací datovou sadou se většinou pohybuje mezi 15 a 20. Používané hardwarové vybavení nejúspěšnějších řešitelů většinou spočítá z několika výkoných grafických karet (4 x 1080Ti, 2 x 1080Ti, 4 x 1080Ti, 2 x Titan X, 3 x 1080Ti).

2.3 ImageNet

ImageNet[2] je databáze fotografií vytvořená pro účely výzkumu klasifikace obrazu. Obsahuje přes 14 milionů obrázků, které byly všechny manuálně zařazeny do odpovídající kategorie, aby byla zaručena jejich správnost. Databáze je volně přístupná každému, kdo má zájem o automatické zpracování obrazu. Vzhledem k tomu, že k učení klasifikátorů je potřeba velké množství dat, databáze díky svému rozsahu a kvalitě podstatně napomohla rozvoji nejen rozpoznávání obrazu, ale i hlubokého učení celkově.

[2] http://www.image-net.org/

V rámci projektu ImageNetu navíc probíhá tzv. *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), volně přeložená jako *Výzva ImageNetu pro vizuální rozpoznávání ve velkém měřítku*. Jedná se o soutěž konající se každoročně již od roku 2010, která zahrnuje lokalizaci, klasifikaci a detekci objektů na obrázcích. Již po prvním ročníku soutěže došlo ke znatelnému zlepšení v automatické klasifikaci a s každým rokem dochází k dalšímu postupu. V roce 2012 byl zaznamenám velký úspěch s použitím hlubokých konvolučních neuronových sítích, což mělo za důsledek rozšíření umělé inteligence v komerční sféře. Zapojení společnosti Google do soutěže v roce 2014 pak vedlo k dalšímu zlepšení a také vývoji nového typu konvoluční sítě.

Díky tomu, že ILSVRC je převážně akademickou záležitostí, někteří z účastníků po soutěži zveřejnili své modely, a to včetně vah získaných trénováním na ImageNet datasetu. Část z nich byla dokonce přímo zakomponována do knihoven pro práci s neuronovými sítěmi. To umožňuje všem vytvořit si velmi snadno vlastní klasifikátor, aniž by museli stavět vlastní hlubokou konvoluční síť. S takto před-trénovanými sítěmi je pak navíc možné dále pracovat a například je doučit na vlastních datech. To umožňuje výrazné urychlení trénovacího procesu, neboť kompletní učení velkých sítí, jako jsou tyto, zabere, i při použití výkonného grafického procesoru, stále alespoň několik týdnů.

Kapitola 3

Konvoluční neuronové sítě

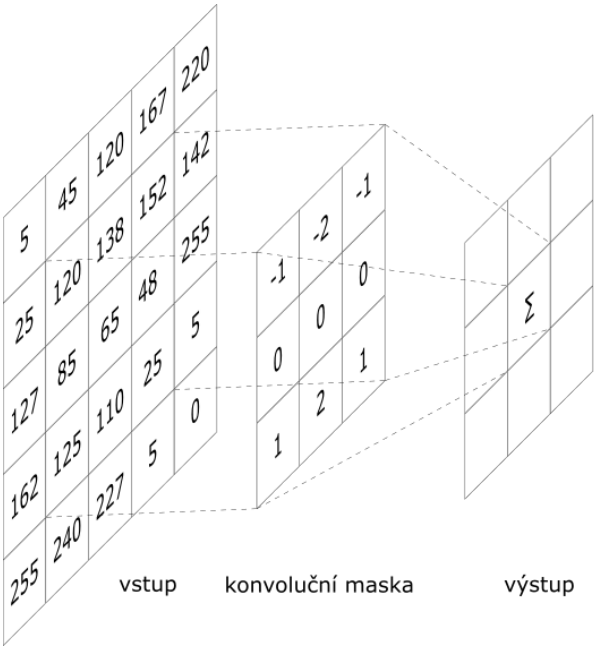
Umělé neuronové sítě[23] (*artificial neural networks*) jsou jednou z podoblastí strojového učení. Jejich vznik byl inspirován snahou napodobit funkčnost lidského mozku, aby mohly řešit předložené problémy stejným způsobem, jako by k nim přistupoval člověk. Pro typy problémů zabývající se interpretací složitých reálných dat z různých snímačů (tedy mimo jiné i prací s obrázky) jsou dnes neuronové sítě mezi nejefektivnějšími dostupnými metodami řešení.

Modely neuronových sítí vycházejí ze struktury nervového systému živých organismů. Tak jako je nervová soustava tvořena složitou sítí vzájemně propojených nervových buněk – neuronů, tak jsou i umělé neuronové sítě složeny ze vzájemně propojených elementů – umělých neuronů.

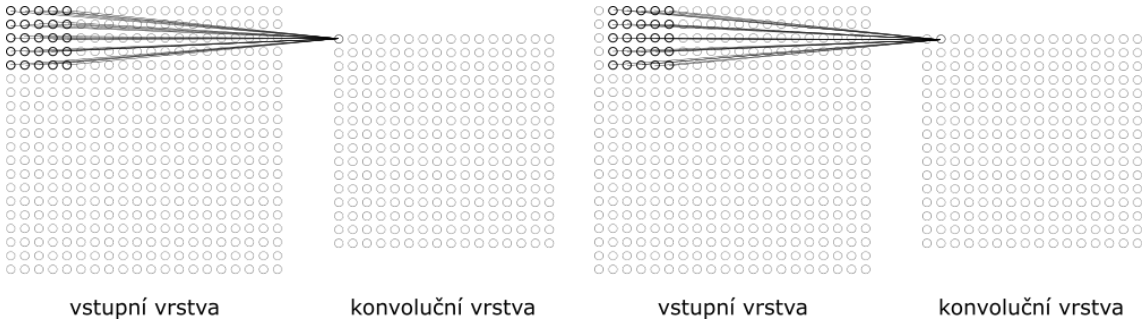
Umělý neuron můžeme rozdělit na několik částí – vstupní část, vlastní tělo neuronu, blok aktivační funkce a výstup. Obecně má n reálných vstupů $x_1 - x_n$. Ty jsou ohodnoceny odpovídajícími synaptickými váhami $w_1 - w_n$, které určují vliv jednotlivých vstupů na výstup. V těle neuronu se spočítá součet vstupních hodnot násobených příslušnými váhami a od výsledné hodnoty je následně odečten práh. Takto získaná hodnota se nazývá vnitřní potenciál neuronu. Potenciál neuronu dále vstupuje do bloku aktivační funkce, který rozhoduje o výstupní hodnotě neuronu. Aktivační přenosová funkce je obecně nelineární funkce transformující hodnotu potenciálu. Jako aktivační funkce se používají například sigmoida nebo hyperbolický tangens.

Neuronová síť vzniká propojením neuronů tak, že výstup neuronu je vstupem obecně více neuronů, podobně, jako je to u biologických neuronů. Celkový počet a vzájemné propojení neuronů v síti určují tzv. *architekturu* (nebo také topologii) sítě. Neurony můžeme rozlišit na vstupní, výstupní a skryté. Vstupní neurony nemají žádné vstupní synapse, ani práh a přenosovou funkci, jejich výstupy představují vstupy pro celou síť. Výstupy výstupních neuronů pak představují výstupy celé sítě. Skrytými nazýváme neurony, které nejsou ani vstupní ani výstupní, tedy leží někde mezi nimi, a vytváří cesty ze vstupu na výstup. Šíření a zpracování informace na cestě v síti je umožněno změnou stavu neuronů ležících na této cestě. Stavy všech neuronů v síti určují tzv. *stav* sítě a synaptické váhy všech spojů v síti představují tzv. *konfiguraci* sítě.

Konvoluční neuronové sítě jsou speciálním druhem vícevrstvých neuronových sítí, jejichž vznik byl inspirován studií o zrakovém systému koček[8]. Jsou tedy přímo uzpůsobeny práci s obrázky a videi, dokážou v nich rozpoznávat vzory přímo a nepotřebují žádné zvláštní předzpracování vstupních dat.



Obrázek 3.1: Ukázka dvourozměrné diskrétní konvoluce.



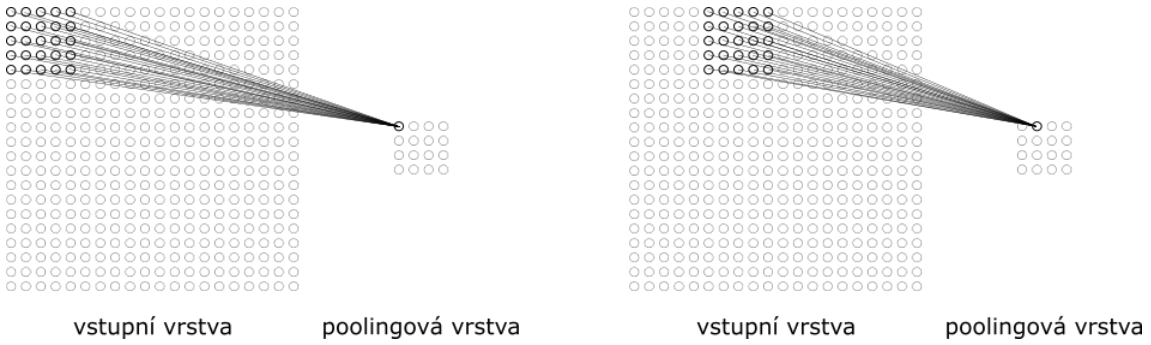
Obrázek 3.2: Ukázka propojení neuronů v konvoluční vrstvě.

3.1 Architektura konvolučních neuronových sítí

V architekturách konvolučních sítí se používají tři typy vrstev – *konvoluční vrstva* (*convolutional layer*), *poolingová vrstva* (*pooling layer*) a *plně propojená vrstva* (*fully-connected layer*).

Konvoluční vrstva Konvoluční vrstva aplikuje na vstupní data matematickou operaci konvoluce[1], konkrétně se jedná o dvourozměrnou diskrétní konvoluci. Vstupní obrázek je reprezentován dvourozměrnou maticí pixelů, z nichž každý má hodnotu v rozsahu $0 - 255$ (intenzita pixelu). Jádro konvoluce je potom tabulka (konvoluční maska), kterou přiložíme k obrázku a spočítáme hodnotu konvoluce pro překrytou oblast: jednoduše vynásobíme hodnotu každého pixelu v oblasti hodnotou masky, která mu překrytím odpovídá, a výsledné hodnoty sečteme. Tím dostaneme novou hodnotu pixelu ležícího uprostřed dané oblasti. Masku poté posuneme na další pozici a proces opakujeme, dokud neprojdeme celou plochu

[1] https://cs.wikipedia.org/wiki/Konvoluce



Obrázek 3.3: Ukázka propojení neuronů v poolingové vrstvě.

obrázku. Při práci s barevným obrázkem počítáme konvoluci pro každou barevnou složku zvlášť.

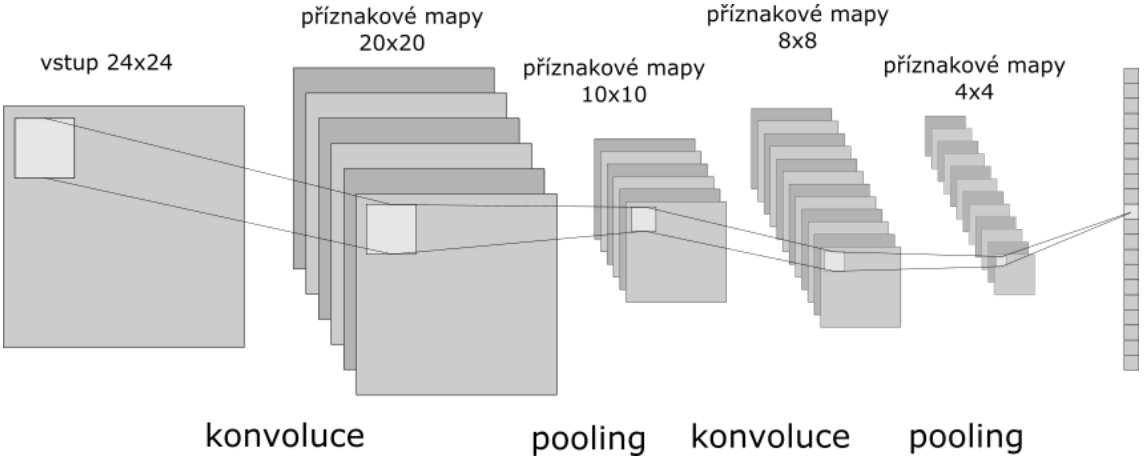
U neuronových sítí je konvoluce implementována pomocí hierarchického zapojení neuronů. Každý neuron v konvoluční vrstvě neuronové sítě je propojen s malou oblastí neuronů, tzv. *recepční pole* (*receptive field*) neuronu, v předchozí vrstvě, přičemž pro sousední neurony se tyto pole částečně překrývají, jak můžeme vidět i na obrázku 3.2. Konvoluční masku zde tedy tvoří synaptické váhy spojů. Vzhledem k tomu, že všechny neurony ve vrstvě plní stejnou funkci (snaží se detekovat stejný příznak/stejné příznaky), pouze nad jinou podmnožinou vstupních dat, používají všechny jednu sdílenou sadu hodnot vah spojů a prahu. Všechny neuronu v jedné vrstvě jsou tedy totožné, pouze jsou oproti sobě posunuté.

Váhy neuronů nám společně s prahem definují tzv. *filtr* (kernel), po jehož aplikaci můžeme z obrázku získat určité příznaky. Například můžeme detekovat horizontální nebo vertikální hrany. Výstupem po aplikaci filtru je tzv. *příznaková mapa*. Málokdy nám však bude stačit pouze jeden příznak, většinou na obrázek aplikujeme více různých filtrů. Pak dostaneme jednu příznakovou mapu za každý aplikovaný filtr. V případě, že pracujeme s barevným obrázkem, dostaneme také zvláštní příznakovou mapu pro každou barevnou složku. Výstupem konvoluční vrstvy je tedy soubor příznakových map.

Poolingová vrstva Poolingová, nebo také subsaplingová, vrstva většinou následuje za konvoluční vrstvou. Jejím účelem je zmenšit velikost výstupních dat konvoluční vrstvy a tím pádem snížit množství parametrů, které by se musely počítat.

Neurony v poolingové vrstvě jsou poskládány podobně jako neurony v konvoluční vrstvě. Každý neuron je spojený s malou oblastí neuronů v předcházející konvoluční vrstvě, rozdíl je však v tom, že v tomto případě se oblasti spojené s neurony nijak nepřekrývají, viz. obrázek 3.3. Pokud tak ze vstupní matice o rozměrech N x N budeme vybírat podoblasti o rozměrech k x k, výsledná matice bude mít rozměr $\frac{N}{k}$ x $\frac{N}{k}$. Pokud je výstupem konvoluční vrstvy více než jedna příznaková mapa, pak je funkce poolingové vrstvy aplikována na každou mapu zvlášť a výstupem poolingové vrstvy je jedna příznaková mapa za každou vstupní příznakovou mapu.

Existuje několik různých funkcí, které je možné použít v poolingové vrstvě. Asi nejčastěji používanou je tzv. *max-pooling* neboli výběr maxima. Tato funkce vybere ze své vstupní oblasti neuron s největší výstupní hodnotou a tuto hodnotu použije jako svou výstupní. Tato funkce je založená na předpokladu, že pokud ve vstupních datech identifikujeme daný příznak, přesná pozice jeho výskytu už pro nás není tak důležitá, stačí nám jeho přibližná pozice vzhledem k ostatním příznakům.



Obrázek 3.4: Jednoduchá konvoluční neuronová síť.

Plně propojená vrstva V plně propojené vrstvě je vstup každého neuronu propojen s výstupy všech neuronů v přechozí vrstvě. Tato vrstva následuje za poslední poolingovou vrstvou a transformuje její výstupní příznakovou mapu na jednorozměrný vektor.

Shrnutí Konvoluční neuronová síť se tedy skládá z jedné či více konvolučních vrstev, přičemž v naprosté většině případů je každá konvoluční vrstva následována poolingovou vrstvou. Za konvolučními a poolingovými vrstvami pak následuje alespoň jedna plně propojená vrstva. Poslední plně propojená vrstva představuje výstup sítě. Příklad jednoduché konvoluční sítě je na obrázku 3.4

3.2 Učení konvolučních neuronových sítí

Při učení (nejen konvolučních) neuronových sítí se nejčastěji používá metoda zpětného šíření chyby neboli *backpropagation*. Tato metoda byla v podobě, ve které se dodnes používá, představena v roce 1986 Davidem Rumelhartem[17]. Metoda spočívá v úpravě synaptických vah v síti na základě rozdílu mezi výstupem sítě a očekávaným výsledkem. Tento rozdíl nazýváme chybou sítě. Úpravou vah se tuto chybu snažíme minimalizovat, k čemuž se používají gradientní metody.

V praxi můžeme učení sítě rozdělit na dva kroky. V prvním kroku je síti předložen vektor vstupních dat, který síť příslušným způsobem zpracuje, a vrátí nám výstupní vektor. Ten pak porovnáme s očekávaným výstupem a spočítáme chybu E. Standartní používanou chybovou funkcí je tzv. střední kvadratická chyb (*mean squared error*).

Ve druhém kroku chceme spočítat gradient chybové funkce a podíly, které v něm mají jednotlivé váhy neuronů, tedy jejich parciální derivace. Váhy jednotlivých neuronů pak chceme změnit v opačném směru gradientu. Zpětný průchod sítí (tj. ve směru od výstupní vrstvy ke vstupní) nám umožňuje efektivní výpočet parciálních derivací jednotlivých vah pomocí řetízkového pravidla (*chain rule*) pro výpočet složených derivací na sobě závislých funkcí. V obyčejné vícevrstvé neuronové síti pak můžeme chybu j-tého neuronu v k-té vrstvě vyjádřit vzorcem:

$$\delta_j^k = \frac{\partial E}{\partial a_j^k} = \sum_{l=1}^{m} \frac{\partial E}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_j^k} = \sum_{l=1}^{m} \delta_j^{k+1} \frac{\partial a_l^{k+1}}{\partial a_j^k}, \text{ kde } a_j^k = \sum_{l=1}^{n} w_{jl}^k \sigma(a_l^{k-1}) + b_j^k, \qquad (3.1)$$

n je počet vstupů neuronu a m je počet neuronů ve vrstvě $k+1$.

U konvolučních sítí je maticové násobení nahrazené konvolucí, výpočet vnitřního potenciálu neuronu tedy vypadá takto:

$$a_{x,y}^{l+1} = w^{l+1} * \sigma(a_{x,y}^{l}) + b_{x,y}^{l+1} = \sum_{i} \sum_{j} w_{i,j}^{l+1} \sigma(y_{x-i,y-j}^{l}) + b_{x,y}^{l+1}. \tag{3.2}$$

Když tento vztah dosadíme do vzorce 3.1 a upravíme[11][4], dostaneme pak vzorec pro výpočet chyby v konvolučních sítích:

$$\delta_{x,y}^{k} = \delta_{x,y}^{k+1} * rot_{180^\circ}(w_{x,y}^{k+1}) \sigma'(a_{x,y}^{k}). \tag{3.3}$$

Kapitola 4

Použité modely sítí

Hluboké konvoluční neuronové sítě zaznamenávají velké úspěchy při práci s obrázky a momentálně jsou nejlepší existující metodou pro řešení úloh jako je klasifikace. Jejich nevýhodou však stále je výpočetní a časová náročnost učení. Navíc k úspěšnému natrénování sítě je potřeba dostatečné množství tréninkových dat. Hluboké sítě mohou obsahovat až několik milionů parametrů a pokud jim nepředložíme dostatečně velký dataset, dojde velmi snadno k jejich přeučení.

V praxi se tedy jen velmi zřídka trénují hlubší konvoluční sítě od začátku (tedy s náhodným počátečním nastavením vah). Místo toho se využívá technika zvaná *transfer learning*[12]. Ta spočívá v před-trénování sítě na dostatečně velkém datasetu (nejčastěji se využívá již dříve zmíněný ImageNet) a následném dotrénování (tzv. *finetuning*) na konkrétním datasetu. Alternativně se před-trénovaná síť dá použít jako extraktor vlastností (*feature extractor*) vstupních dat.

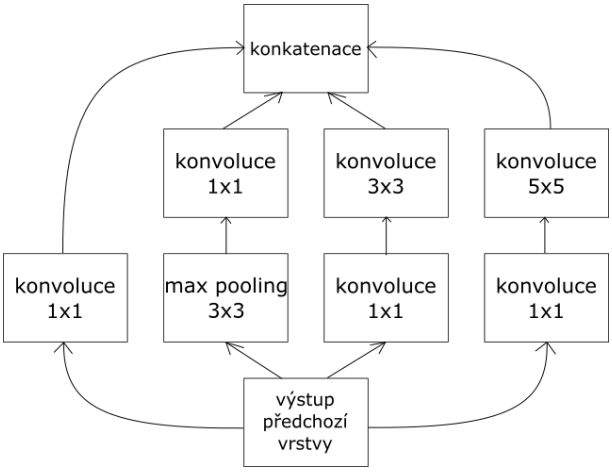
Tento přístup je obzvláště vhodný, pokud máme k dispozici jen omezený tréninkový dataset a natrénování velké sítě jen na něm by nebylo možné. Může nám však pomoct i v případě, kdy máme dostatečné množství dat. Před-trénovaná síť je již schopná rozpoznávat relevantní vlastnosti vstupních dat a její využití nám může výrazně urychlit proces učení. Pokud jsou tedy naše data alespoň částečně podobná datům, na nichž byla síť před-trénovaná, bývá většinou tato metoda využívána.

V našem případě se snažíme klasifikovat fotografie reálných produktů, využití sítě již před-trénované na ImageNet datasetu se tedy přímo nabízí. V této práci byly využity hlavně tři modely před-trénovaných hlubokých konvolučních sítí: *Inception v3*, *Inception-ResNet v2* a‘*MobileNet*.

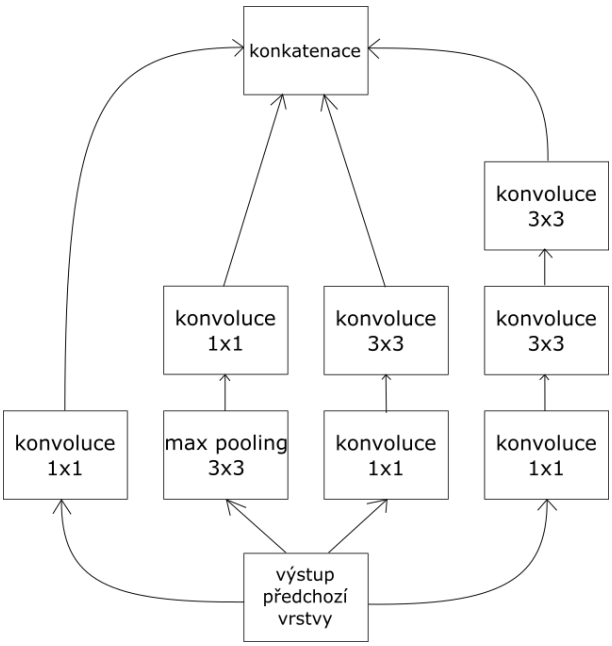
4.1 Inception v3

Po první ukázce potenciálu konvolučních neuronových sítích, byla celkem logickým dalším krokem snaha dosáhnout ještě lepších výsledků přidáním dalších vrstev a prohloubením sítí. Přestože tento přístup se ukázal jako funkční a vznikly například velmi úspěšné modely VGG16 a VGG19, se zlepšením úspěšnosti takovýchto sítí se také velmi výrazně zvyšuje množství parametrů a tím i výpočetní náročnost sítí a jejich náchylnost k přeučení.

Společnost Google se tedy při vývoji své konvoluční sítě obrátila jiným směrem. Místo plně propojených architektur využívají jen částečně propojené, a to i uvnitř konvolucí. Jak ve své práci[1] ukázal Sanjeev Arora, optimální architektura sítě takovéto řídce propojené neuronové sítě může být vystavěna vrstvu po vrstvě analýzou vztahů mezi aktivacemi neu-

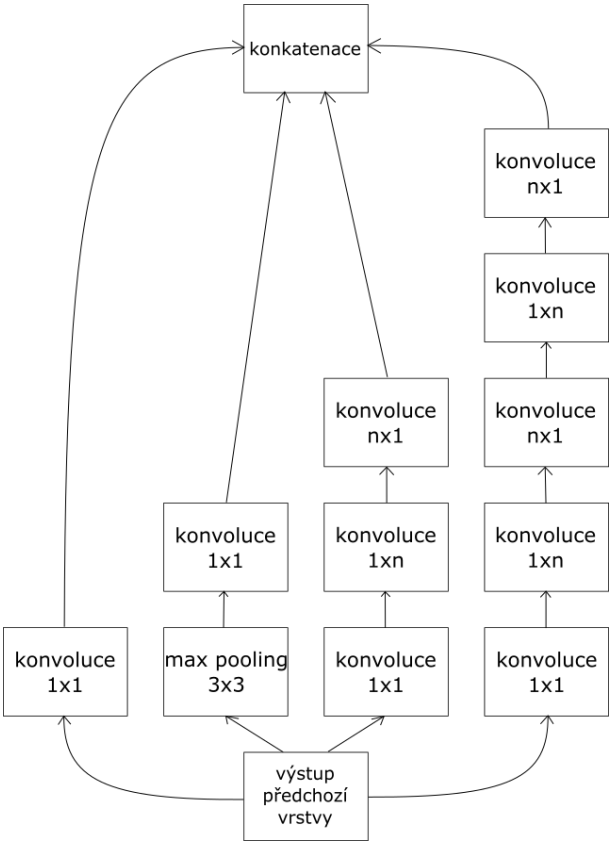


Obrázek 4.1: Základní Inception modul.

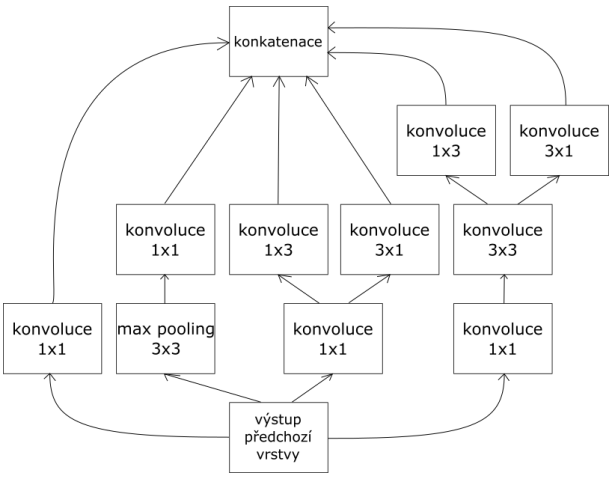


Obrázek 4.2: Upravený Inception modul, ve kterém jsou konvoluce typu 5x5 nahrazeni dvojicemi konvolucí typu 3x3.

ronů z poslední vrstvy a shlukováním neuronů s vysoce provázanými výstupy. Tyto shluky nám pak tvoří jednotky následující vrstvy. Předpokládáme, že každá jednotka z předcházející vrstvy odpovídá určité oblasti obrázku. V nižších vrstvách sítě (blíže ke vstupní vrstvě) bude velké množství shluků koncentrováno ve stejné oblasti a mohou být pokryty konvolucí typu 1x1[14]. Kromě toho dostaneme také menší množství od sebe vzdálenějších shluků, které mohou být pokryty konvolucemi nad většími oblastmi. V Inception sítích jsou (kromě 1x1 konvoluce) využívané konvoluce typu 3x3 a 5x5. Konvoluce přes větší oblasti se nepoužívají, se zvětšením oblasti se totiž zvětšuje i náročnost výpočtu (obecně se konvoluce větší než 5x5 v neuronových sítích příliš nepoužívají).



Obrázek 4.3: Upravený Inception modul využívající asymetrických konvolucí.



Obrázek 4.4: Upravený Inception modul.

Architektura Inception sítě je tedy kombinací těchto tří jmenovaných konvolucí, jejichž výsledky jsou následně spojené do jediného výstupního vektoru tvořícího vstup následující vrstvy. Také se ukázalo jako výhodné přidat k nim i operaci poolingu. Spojením všech těchto operací dostaneme základní stavební jednotku sítě, tzv. Inception modul.

Problémem těchto modulů je, že i malé množství 5x5 konvolucí je neúměrně náročné, pokud navazují na konvoluční vrstvu s větším množstvím filtrů. Z tohoto důvodu jsou před

typ vrstvy	velikost vstupu
3x3 konvoluce	299x299x3
3x3 konvoluce	149x149x32
3x3 konvoluce	147x147x32
3x3 pooling	147x147x64
3x3 konvoluce	73x73x64
3x3 konvoluce	71x71x80
3x3 konvoluce	35x35x192
3 x Inception A (podle obrázku 4.2)	35x35x288
5 x Inception B (podle obrázku 4.3)	17x17x768
2 x Inception C (podle obrázku 4.4)	8x8x1280
8x8 pooling	8x8x2048
plně propojená	1x1x2048

Tabulka 4.1: Architektura sítě Inception v3.

náročnými 3x3 a 5x5 konvolucemi vždy aplikovány ještě konvoluce 1x1, které zmenšují rozměry a usnadňují výpočet. Přidáním těchto 1x1 konvolucí dostaneme kompletní Inception modul, který je zobrazen na obrázku 4.1.

Inception síť je obecně jakákoliv síť skládající se z takovýchto modulů. Moduly však nejsou využity hned v nejnižších vrstvách sítě, ty jsou tvořeny klasickými konvolučními a poolingovými vrstvami, a Inception moduly následují až ve vyšších vrstvách. Toto není nutné pravidlo, pouze se to ukázalo efektivnější z hlediska využití paměti.

První Inception sítí byl GoogLeNet[20], představený v roce 2014 na soutěži ILSVRC. Další verze sítí už jsou pak pojmenované jednoduše Inception vN, kde N je pořadové číslo příslušné verze sítě. Inception v3 je tedy třetí verzí.

V této verzi jsou použity tři verze Inception modulu. V první z nich byla oproti základnímu modulu konvoluce typu 5x5 nahrazena dvěma následnými 3x3 konvolucemi, jak je vidět z obrázku 4.2. Tato změna vede ke snížení náročnosti výpočtu o více než čtvrtinu, aniž by tím došlo ke snížení úspěšnosti sítě.

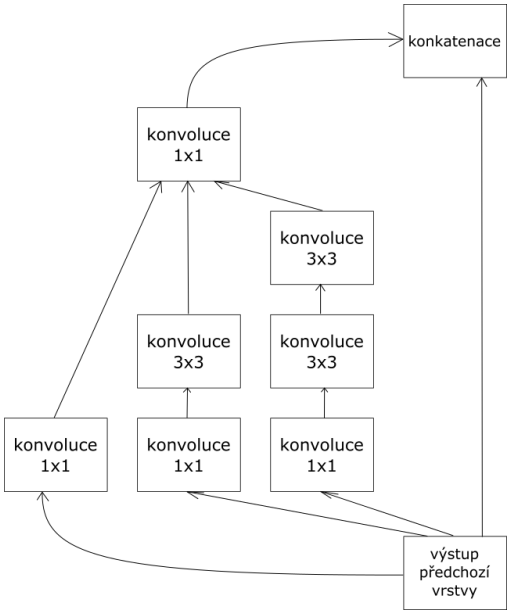
Stejně jako je možné nahradit konvoluci 5x5 konvolucemi 3x3, bylo by možné nahradit konvoluci 3x3 konvolucemi 2x2. Ještě lépe však funguje použití asymetrických konvolucí, tedy konvoluce 3x3 by byla nahrazena konvolucí 3x1 následovanou konvolucí 1x3. Tím dojde ke snížení náročnosti výpočtu o další třetinu. Tento postup je možné využít u všech konvolucí, tedy každá konvoluce n x n může být nahrazena konvolucemi nx1 a 1xn a dojde tím k úspoře na výpočtu. Touto úpravou dostaneme druhý použitý Inception modul, který je vidět na obrázku 4.3.

Třetí verze modulu je ukázána na obrázku 4.4. Je použita pouze na nejhrubší matici obrázku (o velikosti 8x8) a dokáže rychleji zpracovat vícedimenzionální vstup.

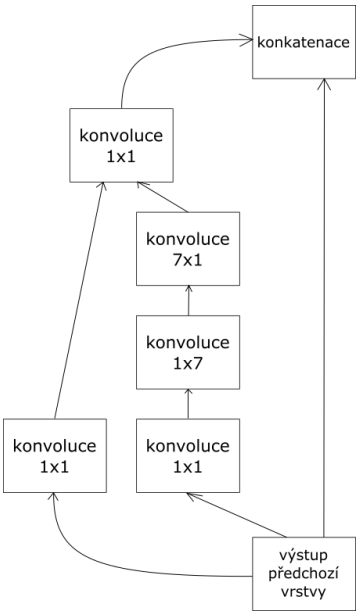
Celková architektura Inception v3 sítě je popsána v tabulce 4.1. Na začátku máme nejdříve klasické konvoluční vrstvy, poté následuje třikrát za sebou verze Inception modulu z obrázku 4.2, pak pětkrát verze modulu z obrázku 4.3, dvakrát verze z obrázku 4.4, a nakonec následuje klasifikátor.

4.2 Inception-ResNet v2

Jak již bylo zmíněno dříve, hlavním principem vývoje tradičních architektur konvolučních neuronových sítí je především přidávání dalších vrstev a prohlubování sítí. Čím hlubší je



Obrázek 4.5: Inception-ResNet modul A.

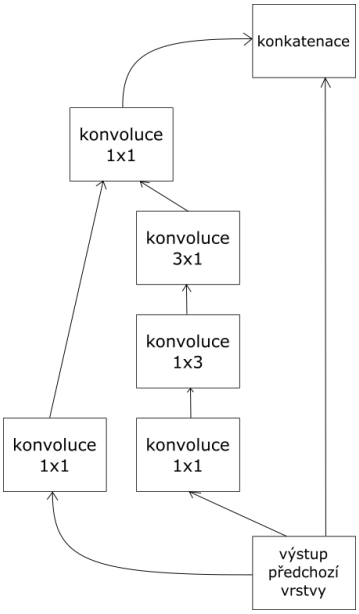


Obrázek 4.6: Inception-ResNet modul B.

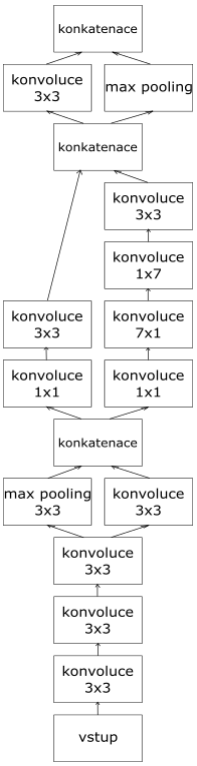
však síť, tím těžší je její trénování. Navíc od určitého bodu začne docházet i k poklesu úspěšnosti. Společnost Microsoft, která při práci s ImageNet datasetem použila sítě s více než 100 vrstvami, tento problém řešila pomocí tzv. *zbytkového učení* (*residual learning*).

Hlavní myšlenkou tohoto je učení je vložení zkratek mezi vrstvy, tedy vstup nižší vrstvy je poskytnut i vyšší vrstvě. Tím se vyřeší problém s klesáním úspěšnosti sítě, a navíc to vede ke snadnějšímu učení sítě.

S použitím tohoto přístupu se Microsoftu podařilo v roce 2015 se sítí zvanou *ResNet*[6] zvítězit v soutěži ILSVRC a zbytkové učení celkově ukázalo velkou užitečnost v hlubokých



Obrázek 4.7: Inception-ResNet modul C.



Obrázek 4.8: Vstupní část (nejnižší vrstvy) sítě Inception-ResNet v2.

konvolučních sítích. Z tohoto důvodu se jej společnost Google pokusila zakomponovat do nové verze své Inception sítě[19]. Vedle *Inception v4* tak tedy zároveň vznikly i *Inception-ResNet v1* a *Inception-ResNet v2*.

typ vrstvy	velikost vstupu
vstupní vrstvy (podle obrázku 4.8)	299x299x3
5 x Inception-Resnet A (podle obrázku 4.5)	35x35x256
redukce A	35x35x256
10 x Inception-ResNet B (podle obrázku 4.6)	17x17x896
redukce B	17x17x896
5 x Inception-ResNet C (podle obrázku 4.7)	8x8x1792
8x8 pooling	8x8x1792
dropout	1x1x1792
plně propojená	1x1x1792

Tabulka 4.2: Architektura sítě Inception-ResNet v2.

Inception-ResNet v2 náročností odpovídá síti Inception v4 a vychází i z její architektury. Liší se pouze v Inception modulech, které jsou nahrazeny novými Inception-ResNet moduly. Moduly v síti Inception v4 zůstávají oproti předchozí verzi Inception v3 v podstatě stejné. Inception-ResNet moduly oproti nim neobsahují operaci poolingu a za každým Inception blokem následuje jedna 1x1 konvoluce bez aktivační funkce, aby se výstup bloku rozměrově vyrovnal vstupu, který se k němu přidává, viz obrázky 4.5, 4.6 a 4.7.

Když nezapočítáme rozdíly v Inception/Inception-ResNet modulech, architektura Inception-Resnet v2 sítě je stejná jako architektura sítě Inception v4. Oproti architektuře sítě Inception v3 se liší především ve vstupní části (nejnižších vrstvách) sítě, kterou máme ukázanou na obrázku 4.8. Celková architektura sítě je pak ukázána v tabulce 4.2. Výstup vstupní části má velikost 35x35, první redukce jej zmenší na 17x17, druhá redukce pak na 8x8.

Na ImageNet datasetu dosáhly sítě Inception v4 a Inception-ResNet v2 zhruba stejné úspěšnosti. Použití residuálních spojení v síti Inception-Resnet v2 však výrazně zvýšilo rychlost učení sítě.

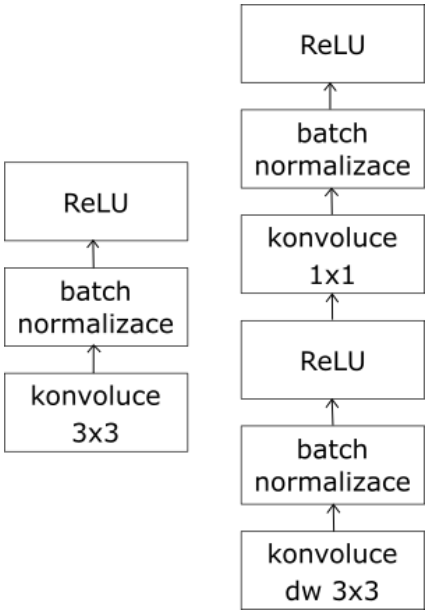
4.3 MobileNet

Prohlubování a celkově zvětšování konvolučních sítí sice vede k vyšší úspěšnosti sítí, ale ne vždy je to vhodné řešení. Tyto sítě, i když mají dobré výsledky, pak nemusí být ve spoustě úkolů použitelné, neboť jsou pomalé a vyžadují velký výpočetní výkon. K reálnému využití sítí však také potřebujeme, aby dokázaly fungovat i na výpočetně omezeném zařízení, které je rozumně dostupné i jinde než v akademickém prostředí nebo ve velkých společnostech. Také pokud mají vykonávat nějakou reálnou činnost musí být schopné reagovat dostatečně rychle.

Z tohoto důvodu Google v roce 2017 představit třídu modelů konvolučních sítí s názvem MobileNet[7], která byla vyvinuta s důrazem na efektivitu. Tyto sítě jsou velmi malé, rychlé a vzhledem k tomu až překvapivě přesné.

Většina pokusů[10, 9, 22] o vytvoření malé, a přitom efektivní, konvoluční sítě spočívá buď ve snaze zkomprimovat již před-trénovanou síť nebo v přímém natrénování malé sítě. MobileNet sítě obsahují dva globální hyper-parametry, které umožňují optimalizovat podobu sítě přímo danému úkolu/dostupným výpočetním prostředkům. Dají se tedy snadno vyladit podle toho, zda chceme dát důraz na úspěšnost nebo na nízkou výpočetní náročnost.

Implementace sítí spočívá v rozdělení konvolucí ve směru hloubky (*depthwise separable convolutions*). Místo standartní konvoluce 3x3 konvoluce zde nejdříve aplikujeme 3x3 filtr na každý vstupní kanál a poté aplikujeme 1x1 konvoluci, abychom spojili získané výstupy.



Obrázek 4.9: Schéma použití batch normalizace a ReLU aktivační funkce v síti MobileNet.

typ vrstvy	velikost vstupu
3x3 konvoluce	224x224x3
3x3 dw konvoluce	112x112x32
1x1 konvoluce	112x112x32
3x3 dw konvoluce	112x112x64
1x1 konvoluce	56x56x4
3x3 dw konvoluce	56x56x128
1x1 konvoluce	56x56x128
3x3 dw konvoluce	56x56x128
1x1 konvoluce	28x28x128
3x3 dw konvoluce	28x28x256
1x1 konvoluce	28x28x256
3x3 dw konvoluce	28x28x256
1x1 konvoluce	14x14x256
5x 3x3 dw konvoluce	14x14x512
5x 1x1 konvoluce	14x14x512
3x3 dw konvoluce	14x14x512
1x1 konvoluce	7x7x512
3x3 dw konvoluce	7x7x1024
1x1 konvoluce	7x7x1024
7x7 pooling	7x7x1024
plně propojená	1x1x1024

Tabulka 4.3: Architektura sítě MobileNet.

Při standartní konvoluci obě tyto akce probíhají v jednom kroku, jejich rozdělením se však velmi výrazně zjednoduší výpočet.

Architektura MobileNet sítí je tvořena takto rozdělenými 3x3 konvolucemi, s výjimkou první vrstvy, ve které je použita standartní konvoluce. Každá konvoluční vrstva v síti je následována batch normalizací a ReLU aktivační funkcí, podle schématu na obrázku 4.9. Kompletní struktura MobileNet sítě je pak vidět v tabulce 4.3.

Přestože architektura MobileNet sítě je už v základu malá, může se stát, že v některých konkrétních případech bude potřeba model, který je ještě menší a rychlejší. Z tohoto důvodu síť obsahuje parametr, který je nazván multiplikátor šířky (*width multiplier*), který umožňuje takovýto menší a méně náročný model vytvořit. Toho dosahuje pomocí zmenšení šířky jednotlivých vrstev sítě. V základním modelu je hodnota tohoto parametru rovna 1, další standartní hodnoty nastavení parametru jsou 0.25, 0.5 a 0.75.

Druhým parametrem sítě je tzv. multiplikátor rozlišení (*resolution multiplier*). Tento parametr snižuje výpočetní náročnost sítě změnou rozměrů vstupního obrázku. Tím dochází ke zmenšení vnitřní reprezentace jednotlivých vrstev.

I přesto, že MobileNet sítě jsou relativně malé a dávají velký důraz na rychlost, základní verze sítě se téměř dokáže vyrovnat větším konvolučním sítím (například VGG16, oproti které má 32krát méně parametrů). Zmenšené varianty sítě pak logicky dosahují menší úspěšnosti, přesto jsou však stále relativně úspěšné.

Kapitola 5

Implementace

5.1 Použité nástroje

Tensorflow Existuje několik různých frameworků pro práci s neuronovými sítěmi. Příkladem můžou být třeba MXNet[1], Microsoft Cognitive Toolkit[2], Caffe[3] nebo Tensorflow[4]. V této práci byl použit poslední jmenovaný. Tensorflow je open source knihovna vyvinutá týmem Google Brain [5] pro interní potřeby společnosti. V září 2015 pak byla zveřejněna pod Apache licencí.

Tensorflow umožňuje provádění vysoce náročných výpočtů jak na standartních, tak i na grafických procesorech. Také je s jeho pomocí možné provádět výpočty na několika procesorových jednotkách zároveň. Při použití grafických procesorů běží Tensoflow na platformě CUDA[6] a podporuje knihovnu cuDNN[7] pro optimalizaci výpočtů.

Keras Keras[8] je vysokoúrovňové API implementované v Pythonu, které běží nad Tensorflow (nebo i Theano či Microsoft Cognitive Toolkit) frameworkem. Byl vyvinutý za účelem umožnění jednoduchého a rychlého experimentování s neuronovými sítěmi. Soustředí se především na uživatelskou přívětivost, modularitu a snadnou rozšiřitelnost. Nabízí vyšší úroveň abstrakce než samotný Tensorflow a je velmi intuitivní.

To však neznamená, že Tensorflow není použitelný samotný. Součástí Tensorflow jsou mimo jiné i oficiální API pro jazyky Python a C, navíc Tensorflow pracuje na nižší úrovni abstrakce a umožňuje nám tedy více zásahů do výpočtů a lepší kontrolu nad sítěmi. Avšak vzhledem k tomu, že v této práci nebylo potřeba pracovat na nižší úrovni abstrakce, byl v ní použit Keras běžící nad Tensorflow.

Metacentrum MetaCentrum, které existuje již od roku 1996, je součástí aktivit sdružení CESNET[9]. Věnuje se především provozu a rozvoji gridové infrastruktury v České republice, konkrétně se zabývá budováním národního Gridu a jeho propojením se souvisejícími

[1] https://mxnet.apache.org/
[2] https://www.microsoft.com/en-us/cognitive-toolkit/
[3] http://caffe.berkeleyvision.org/
[4] https://www.tensorflow.org/
[5] https://en.wikipedia.org/wiki/Google_Brain
[6] https://developer.nvidia.com/cuda-zone
[7] https://developer.nvidia.com/cudnn
[8] https://keras.io/
[9] https://www.cesnet.cz/

mezinárodními aktivitami. Má statut Národní Gridové Infrastruktury (*National Grid Infrastructure*, NGI) České republiky v rámci Evropské Gridové Infrastruktury[10] (*European Grid Infrastructure*, EGI).

MetaCentrum umožňuje bezplatné členství, a tedy přístup ke své výpočetní kapacitě, všem akademickým pracovníkům a studentům členů sdružení CESNET. Výpočetní kapacita MetaCentra obsahuje (mimo jiné) i dva clustery se stroji určenými pro výpočty na grafickém procesoru podporujícími platformu CUDA a knihovnu cuDNN. Každý z těchto strojů je vybaven dvěma grafickými kartami nVidia Tesla K20. Téměř všechny výpočty potřebné v rámci této práce (tj. trénování a testování modelů sítí) byly provedené na strojích MetaCentra.

5.2 Příprava dat

Jak bylo zmíněno v kapitole 2, testovací data poskytnutá do soutěže bohužel neobsahují identifikátory kategorií. V této práci tedy máme k dispozici pouze trénovací dataset. Abychom však byli schopní vyhodnotit úspěšnosti jednotlivých modelů, potřebujeme nějaká testovací data. Naštěstí je náš dostupný trénovací dataset dostatečně velký, aby z něj bylo možné část produktů odebrat a použít je při vyhodnocování úspěšnosti modelů, a přitom nám stále zůstalo dostatečně velké množství dat pro natrénování modelů.

Ideální rozdělení by bylo takové, aby nově vytvořená validační sada co nejvíce napodobovala původní testovací sadu. Vzhledem k tomu, že však o testovací sadě téměř nic nevíme a nejsme schopni zjistit rozložení produktů mezi kategoriemi, nemůžeme z ní ani vycházet při vytváření validační sady. Dá se však předpokládat, že v množství poskytnutých dat se případné rozdíly částečně ztratí.

Když se podíváme alespoň na počty produktů v obou původních sadách, zjistíme že zhruba čtyři pětiny patří do trénovací sady a jedna pětina do testovací. Toto se zdá jako poměrně rozumné rozdělení, budeme se ho tedy držet.

Data jsou v původním souboru uložená ve formátu binárního jsonu, přičemž obrázky jsou zde zkomprimované. I tak ale tento soubor zabírá více než 60 GB paměti. Kdybychom z něj obrázky vytáhli, a uložili je v nezkomprimované podobě, potřebovali bychom odhadem 380 GB paměti. Rychlost načítání obrázků do operační paměti by se nám přitom zvýšila jen velmi málo, vzhledem k předpokládané výpočetní náročnosti trénování vybraných modelů sítí bychom si tím nijak nepolepšili. Nově vytvořené datové sady tedy budou uložené ve stejném formátu jako původní data, abychom ušetřili úložný prostor.

Rozdělení dat na trénovací a validační sadu probíhalo ve dvou fázích. V první fázi byla nejdříve projita data a zjištěny celkové počty produktů v jednotlivých kategoriích. Přitom byl pro každou kategorii vytvořen list identifikátorů produktů. Produkty v listech byly náhodně zamíchány a rozděleny tak, aby v trénovací sadě byly čtyři pětiny produktů z každé kategorie a ve validační sadě zbytek. Poté bylo zapotřebí znovu projít původní soubor a tentokrát z něj vzít položky i s obrázky. Před uložením do nových souborů pak byly nově vzniklé sady znovu náhodně zamíchány.

5.3 Práce s daty

Po rozdělení dat na dvě části dostaneme trénovací sadu s 5 658 015 produkty (celkově 9 899 092 obrázků) a validační sadu s 1 411 881 produkty (celkem 2 472 201 obrázků).

[10] https://www.egi.eu/

Je tedy celkem očividné, že data nebude možné naráz načíst do operační paměti. Řešením je procházet postupně soubor s daty a načíst vždy jen data potřebná pro konkrétní krok trénování sítě.

Trénování sítě je však už samo o sobě výpočetně a časově náročné a prokládat ho ještě s načítáním dat by vedlo k dalšímu zpomalení. Ideálně tedy chceme, aby načítání dat probíhalo paralelně s trénováním a nedocházelo ke zbytečným prodlevám. Proto budeme potřebovat vhodnou funkci, která bude mít na starosti načítání dat ze souboru a poběží na svém vlastním vlákně.

Python umožňuje velmi snadnou implementaci takové funkce pomocí iterátoru. Iterátor je objekt, který umožňuje procházet přes položky kolekce nezávisle na konkrétním typu. Tento iterátor je v našem případě implementován v podobě třídy *threadsafe_iter* a kromě umožnění iterace nad položkami zajišťuje také bezpečný přístup k souboru.

S pomocí této třídy jsou pak implementovány generátory, které načtou ze souboru určený počet položek, převedou je do potřebného formátu a předají je do hlavního vlákna. Základní generátor vezme z každé položky první obrázek a identifikátor kategorie a vytvoří tak dvě pole. Hodnoty intenzity pixelů obrázků jsou přitom převedeny do intervalu od nuly do jedné a hodnoty identifikátorů kategorií jsou transformované na pořadová čísla kategorií (tj. hodnoty do 5 270). Jednu takto vrácenou skupinu položek nazýváme *batch* (dávka).

5.4 Trénování modelů

Než začneme trénovat modely je třeba jim nejdříve inicializovat váhy. Pokud bychom začínali modely trénovat od nuly, inicializovali bychom váhy na náhodné hodnoty. V této úloze však nemusíme trénovat modely od začátku, můžeme použít hodnoty vah získané trénováním modelů na ImageNet datasetu. Tyto hodnoty vah jsou dokonce v Kerasu přímo obsaženy a nemusíme je tedy zvlášť stahovat. Použitím těchto vah ušetříme čas při trénování, neboť síť by již měla být schopná rozeznávat objekty na obrázcích, potřebujeme ji naučit už jen to, jak je správně rozřadit do našich daných kategorií.

Při trénování byly využity dvě optimalizační metody[16] – SGD (stochastický gradientní sestup, *stochastic gradient descent*) a Adam (odhad adaptivního momentu, *adaptive moment estimation*). Stochastický gradientní sestup je nejběžněji používaná optimalizační metoda. Hlavní myšlenkou této metody je, že úprava vah probíhá pomocí nejobvyklejšího gradientu, který získáme jako průměr všech gradientů v dávce. Adam vznikl kombinací dvou dalších optimalizačních metod – metody AdaGrad (adaptivní gradient, *adaptive gradient*) a metody RMSprop. Adam si uchovává exponenciálně klesající průměr předchozích gradientů a exponenciálně klesající průměr druhých mocnin předchozích gradientů, což jsou odhady prvních a druhých momentů gradientů. Pomocí těchto odhadů momentů jsou pak spočítány nové hodnoty vah. Metoda SGD byla použita v kombinaci s momentem, ale přesto se metoda Adam ukázala jako efektivnější a vedla k rychlejšímu učení. SGD tedy bylo použito jen u trénování základních modelů a u experimentů už pak byl využit jen Adam. Hodnota koeficientu učení (*learning rate*) byla nastavována individuálně pro jednotlivé trénované modely.

Chybová funkce byla použita pro všechny modely stejná. Jednalo se o tzv. *sparse categorical cross entropy*, překlad by byl asi řídká kategorická křížová entropie. Funguje stejně jako obyčejná kategorická křížová entropie, liší se podobou očekávaného výstupu. Kategorická křížová entropie očekává výstup jako vektor s velikostí odpovídající počtu výstupů, kde hodnota odpovídající správné kategorii je rovná 1 a ostatní hodnoty jsou nulové. Tedy pokud máme tři kategorie a druhá v pořadí je správná, pak očekávaný výstup by byl vektor

$[0, 1, 0]$. Řídká křížová entropie očekává správný výstup jako číslo odpovídající správné kategorii, v tomto případě by to tedy bylo číslo 2. Chyba se pak počítá jako záporná hodnota logaritmu předpovězené pravděpodobnosti správné kategorie. Oproti klasické střední kvadratické chybě, která dává až zbytečně velký důraz na chybné výstupy sítě, se tato funkce zaměřuje na správný výstup sítě. A vzhledem k tomu, že při klasifikaci se snažíme pro každý vstup získat jeden jednoznačný výstup, tato funkce se zdá být vhodnou volbou.

Modely, které používáme, byly vytvořeny pro zařazování obrázků do jedné z 1 000 kategorií. My však máme kategorií 5 270. Nemůžeme tedy jen vzít modely, tak jak jsou. U konvolučních neuronových sítí však naštěstí není nijak zvlášť složité upravit model pro jiný počet výstupních tříd. Modely totiž fungují tak, že konvoluční vrstvy slouží k detekci různých vlastností obrázku, například v nejnižších vrstvách detekujeme hrany v obrázku, ve vyšších vrstvách zjišťujeme konkrétní tvary objektů a v nejvyšších vrstvách pak získáváme tzv. vysoko-úrovňové vlastnosti (*high-level features*) obrázku, pomocí kterých pak obrázek klasifikujeme. Samotné přiřazení do kategorie, pomocí zjištěných vlastností, ale probíhá až v plně propojené vrstvě, respektive vrstvách, která následuje za konvolučními vrstvami. Keras umožňuje vytvořit modely bez těchto posledních vrstev jednoduchým nastavením parametru *include_top=False* v konstruktoru sítě. K takto vytvořenému modelu pak přidáme své vlastní plně propojené vrstvy, které budou odpovídat našim kategoriím. Někdy bývá vhodné přidat i poolingovou vrstvu a někdy se nám může hodit také dropout[18].

Inception v3 Pokud vytvoříme model sítě Inception v3 bez posledních (klasifikačních) vrstev, přijdeme tím oproti celému modelu o jednu poolingovou vrstvu, konkrétně vrstvu typu *GlobalAveragePooling2D* (v Kerasu), a o jednu plně propojenou vrstvu, v Kerasu se jedná o vrstvu typu Dense. Při přidávání našich vlastních vrstev k takto vytvořenému modelu je tedy vhodné začít vrstvou typu *GlobalAveragePooling2D* a poté teprve přidat plně propojené vrstvy. V naší konkrétní implementaci byly použity dvě vrstvy typu *Dense*, první o šířce 4 096 neuronů s aktivační funkcí *ReLU* a druhá o šířce 5 270 neuronů (počet kategorií) s aktivační funkcí *Softmax*.

Inception-ResNet v2 U sítě Inception-ResNet v2 nastavením parametru *include_top* na hodnotu *False* vytvoříme model, ve kterém bude oproti plnému modelu chybět pouze úplně poslední vrstva, tedy jedna plně propojená vrstva typu *Dense*. Takto vytvořený model přitom končí poolingovou vrstvou typu *GlobalAveragePooling2D*. Naše vlastní přidané vrstvy tedy žádnou poolingovou vrstvu obsahovat nebudou. Přidané plně propojené vrstvy jsou stejné jako u modelu Inception v3, tedy jedna vrstva typu *Dense* o šířce 4 096 neuronů s aktivační funkcí *ReLU* a druhá vrstva typu *Dense* o šířce 5 270 neuronů s aktivační funkcí *Softmax*. Po vyzkoušení dotrénování takto vytvořené sítě se však ukázalo, že má tendenci se velmi rychle začít přeučovat, před každou z těchto dvou *Dense* vrstev tedy byla přidána ještě vrstva typu *Dropout* s koeficientem 0.1.

MobileNet Pro MobileNet síť vypadá klasifikační část modelu trochu složitěji. Skládá se z šesti vrstev, nejdřív je zde *GlobalAveragePooling2D*, ten následují *Reshape* a *Dropout* a poté je zde ještě jedna konvoluční vrstva typu *Conv2D*. Až po těchto čtyřech vrstvách pak následuje plně propojená vrstva s aktivační funkcí *Softmax*. Nakonec následuje další *Reshape*, který převádí výstup do patřičného tvaru. Vzhledem k tomu, že síť MobileNet byla vybraná proto, abychom měli (relativně) rychle se učící síť a zjistili potenciál jednotlivých experimentů v rozumném časovém úseku, nebudeme se zde ponořovat do experimentů s její architekturou a ke klasifikaci použijeme pouze plně propojené vrstvy stejně jako u zbylých

modelů. K modelu sítě bez posledních šesti vrstev tedy přidáme jednu vrstvu typu *GlobalAveragePooling2D* a dvě vrstvy typu *Dense*, jednu o šířce 4 096 s aktivační funkcí *ReLU* a druhou o šířce 5 270 s aktivační funkcí *Softmax*.

Kapitola 6

Experimenty

6.1 Metriky pro vyhodnocení experimentů

Vyhodnocování úspěšnosti je řešeno nejjednodušším možným způsobem. Úspěšnost sítě je vyjádřena v procentech, přičemž procento úspěšnosti je pro nás podíl produktů, které model správně zařadil do kategorie, vůči celku. Stoprocentní úspěšnost by tedy znamenala, že všechny produkty ve validační sadě byly zařazeny do správné kategorie, zatímco nulová úspěšnost by znamenala, že do správné kategorie nebyl zařazen ani jeden z nich.

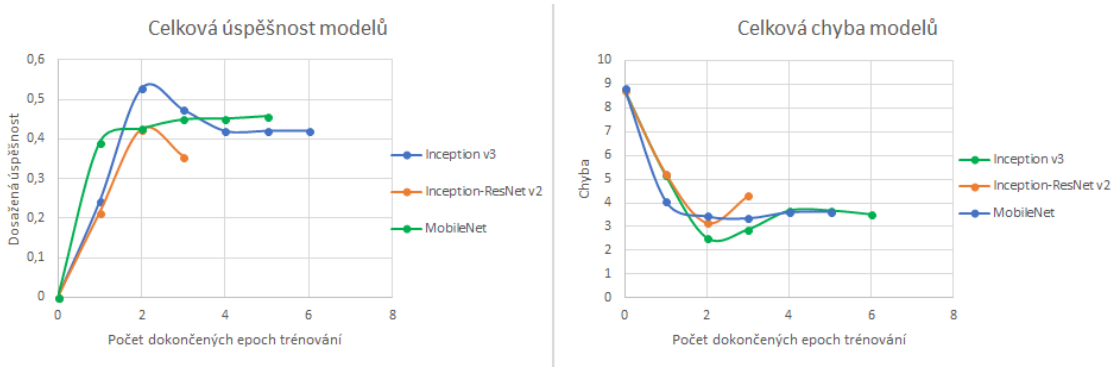
Výslednou kategorii pro daný produkt získáme tak, že po projití vstupních dat sítí vezmeme výsledný vektor pravděpodobností jednotlivých kategorií a vybereme z nich tu s nejvyšší hodnotou. Tu pak porovnáme s očekávanou kategorií a pokud se shodují, prohlásíme tento produkt za úspěšně klasifikovaný. Po projití celého validačního datasetu pak už jen podělíme počet takto úspěšně zařazených produktů celkovým počtem produktů ve validačním datasetu a výsledek převedeme na procenta.

6.2 Základní modely

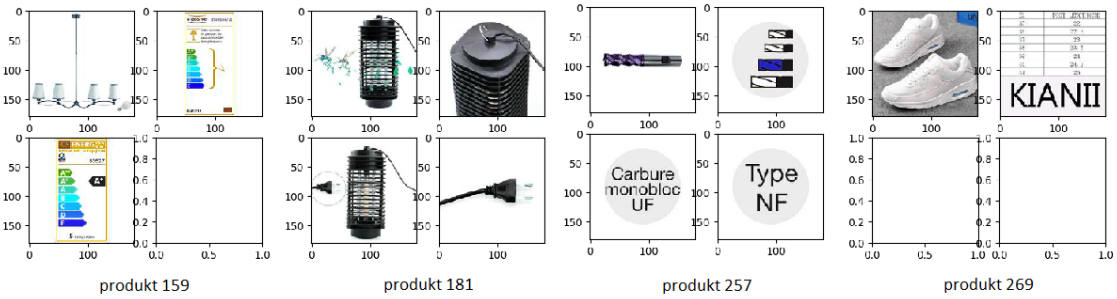
Pokud chceme pomocí experimentů zjistit, jaké přístupy vedou k natrénování úspěšné sítě, potřebujeme nejdřív mít nějaký základ, se kterým budeme moct modely natrénované v rámci experimentů porovnávat. Chceme tedy natrénovat jeden model (respektive jeden model od každého použité typu sítě) bez toho, abychom jakkoliv upravovali vstupní data nebo optimalizovali učení.

Pro každý typ sítě tedy vytvoříme jeden model (viz kapitola 5.4) a ten necháme trénovat na všech obrázcích z trénovacího datasetu. Nijak přitom neřešíme, že některé z obrázků patří ke stejnému produktu. Z praktických důvodů byl pro tyto modely vytvořen zvláštní trénovací a validační dataset, kde jsou položky s více obrázky zaměněny za více položek se stejnými identifikátory, z nichž každá obsahuje pouze jeden obrázek. Tím dojde k malému zrychlení načítání dat z úložiště.

Při trénování sítí Inception v3 a Inception-ResNet v2 byla nejdříve použita optimalizační metoda SGD s koeficientem učení 0,001 a momentem 0,9, která byla v průběhu nahrazena optimalizační metodou Adam s koeficientem učení 0,001, když se ukázalo, že použití optimalizační metody Adam vede k rychlejšímu učení. U sítě MobileNet už byla použita pouze optimalizační metoda Adam. Rozdíl v rychlosti učení těchto dvou metod je vidět již po prvních dvou epochách trénování. V prvních epoše trénování pak docházelo k úpravám vah pouze našich přidaných vrstev, zatímco převzaté modely sítí používaly váhy získané před-



Obrázek 6.1: Celkové úspěšnosti a chyby jednotlivých sítí v průběhu trénování.



Obrázek 6.2: Ukázka produktů s více než jedním obrázkem.

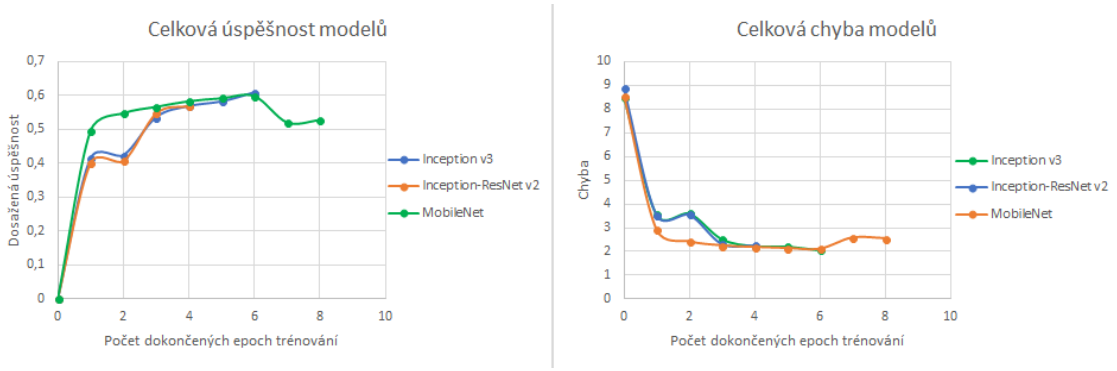
trénováním, aby se vyladil klasifikátor, poté se začal trénovat celý model. Epochou zde (a kdekoliv dále) myslíme jednu kompletní iteraci nad trénovací datovou sadou.

Jak můžeme vidět na obrázku 6.1, sítě se moc dobře netrénují. U modelu Inception v3 dochází ke zlepšování úspěšnosti sítě pouze v prvních dvou epochách. Poté úspěšnost mírně klesne a dále už se nemění. Výsledky modelu MobileNet vypadají velmi podobně pouze s tím rozdílem, že díky použití jiné optimalizační metody začne konvergovat dokonce ještě dřív.

Pokud jde o model Inception-ResNet v2 průběh trénování v prvních dvou epochách vypadá téměř stejně jako u modelu Inception v3. Je zde však velký rozdíl v časové náročnosti učení. Při spuštění trénování sítě na jednom grafickém procesoru nVidia Tesla K20 trvá jedna epocha trénování sítě Inception v3 necelých 39 hodin. Jedna epocha trénování sítě Inception-ResNet v2 však trvá přes 112,5 hodiny, tedy skoro trojnásobek. Dokončení dvou epoch trénování na síti Inception-ResNet v2 tedy zabralo stejně dlouho jako celé trénovaní na síti Inception v3. Již v této fázi tedy můžeme říct, že síť Inception-ResNet v2 pro nás v této práci není vhodná.

Vzhledem k tomu, že nárůst úspěšnosti v prvních dvou epochách byl velmi rychlý a také velmi rychle skončil, můžeme celkem bezpečně usoudit, že sítě se na našem datasetu vůbec netrénují. Dosažená úspěšnost je zde především, pokud ne úplně, důsledkem před-trénování sítí na ImageNetu.

To, že se sítě neučí, však není příliš překvapivé, pokud se podíváme na jednotlivé obrázky v datasetu. Zjistíme totiž, že velmi často pokud produkt obsahuje více než jeden obrázek, pak se produkt se nevyskytuje na všech obrázcích. Některé jeho obrázky mohou obsahovat například příslušenství, detailní záběr jedné součásti produktu nebo doplňující informace.



Obrázek 6.3: Celkové úspěšnosti a chyby jednotlivých sítí v průběhu trénování. Při trénování byl vždy použit pouze jeden náhodně vybraný obrázek od každého produktu.

Ukázku několika takových případů máme na obrázku 6.2. Pokud tedy tyto obrázky jen předložíme síti, nemůžeme se divit, že je není schopná klasifikovat. Navíc celkový počet obrázků patřící produktům s více než jedním obrázkem tvoří téměř dvě třetiny datasetu, takže kdyby polovina z nich neobsahovala produkt, který se snažíme klasifikovat, je to třetina datasetu, na které se bude síť trénovat špatně. V kombinaci s množstvím chyb v datasetu tak nemůžeme příliš doufat, že se síť zvládne něčemu naučit.

Nemůžeme tedy jen trénovat síť na jednotlivých obrázcích, musíme zohlednit jejich příslušnost k produktům. Zde se nabízí dvě možnosti – vybrat z obrázků u produktu vždy jeden, který použijeme při učení nebo zkusit všechny obrázky, které patří ke stejnému produktu, zkombinovat do jednoho obrázku obsahujícího kompletní informace o produktu.

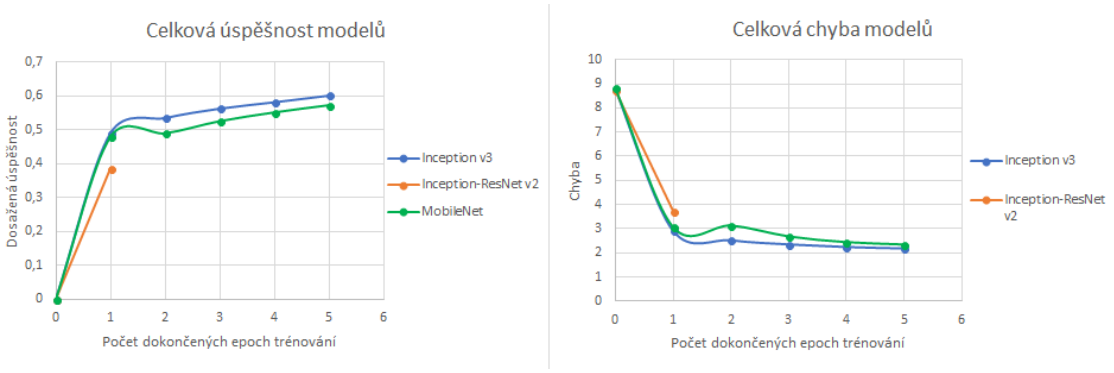
6.3 Učení s náhodným výběrem obrázků

Jak jsme vyvodili z výsledků předchozí části, pokud chceme, aby se nám síť učila, budeme muset zohlednit náležitost jednotlivých obrázků k produktům. Asi nejjednodušším způsobem, jak to udělat, je vybrat vždy od každého produktu jeden obrázek, který následně použijeme při trénování. Bylo by samozřejmě možné vybrat i více než jeden obrázek, ale pak nemáme zaručeno, že tím odstraníme nalezený problém. Můžeme však bezpečně předpokládat, že u každého produktu je alespoň jeden obrázek, ze kterého bude možné produkt klasifikovat. To není úplně stoprocentní pravda, výjimky z tohoto pravidla jsme však již v kapitole 2.2 zahrnuli do očekávané chyby sítí.

Pokud tedy chceme trénovat modely s použitím vždy jednoho obrázku od každého produktu, musíme nejdřív určit, jakým způsobem tento obrázek vybereme. Nejdříve zkusíme použít jednoduchou metodu náhodné volby a natrénovat síť na získaném náhodném výběru obrázků.

Výběr obrázku je implementován ve funkci načítající produkty ze souboru. K tomu je použit jednoduchý generátor náhodných čísel, který vybere číslo z intervalu daného počtem obrázků daného produktu. Toto číslo se pak použije jako index vybraného obrázku, který je následně předán síti k trénování.

Při trénování modelů s použitím náhodného výběru jednoho z obrázků byla použita již jen optimalizační metoda Adam. V prvních dvou epochách nebyly trénované celé modely, ale pouze naše přidané klasifikační vrstvy. Z obrázku 6.3 vidíme, že během druhé epochy se pak přestala zlepšovat jejich úspěšnost a i chybová funkce začala stoupat. Poté tedy bylo



Obrázek 6.4: Celkové úspěšnosti a chyby jednotlivých sítí v průběhu trénování. Při trénování byl od každého produktu vždy použit obrázek s největší pravděpodobností správné kategorie.

puštěno trénování celých modelů. Learning rate byl na začátku nastaven na hodnotu 0,001. Po pěti dokončených epochách byl pak snížen na 0,0001. U modelu MobileNet byl po šesté dokončené epoše snížen ještě na hodnotu 0,00005, ale to už vedlo pouze ke zhoršení úspěšnosti. Model Inception v3 ukazuje ještě další potenciál růstu úspěšnosti, dotrénování sítě až do konce však nebylo možné z časových důvodů. Jedna kompletní epocha nad trénovacími daty vyšla u sítě Inception v3 na něco přes 41 hodin, síť MobileNet byla výrazně rychlejší, jedna epocha zde zabrala pouze něco přes 17 hodin.

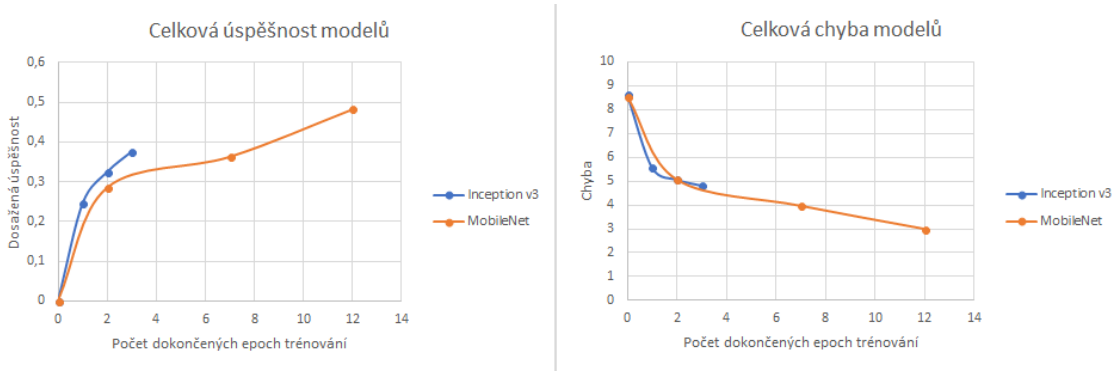
U sítě Inception-ResNet v2 se znovu ukázalo, že pro náš dataset není příliš vhodná. Jedna epocha nad trénovacími daty u ní vyšla zhruba na 100 hodin, tedy skoro 2,5násobek času potřebného pro jednu epochu sítě Inception v3. Přitom dosahuje po stejném počtu epoch v podstatě stejných výsledků.

Jak je tedy vidět omezením trénování vždy jen na jeden obrázek od každého produktu se nám podařilo zvýšit úspěšnost modelů. Největší zlepšení zaznamenala síť Inception v3, u které se zvýšila úspěšnost o více než 18 %. Přitom jsme nijak nezaručili, že obrázek vybraný pro trénování bude vhodný, dá se říct, že jsme pouze podstatně snížili počet nevhodných obrázků.

6.4 Učení s výběrem nejúspěšnějšího obrázku

V předchozí části jsme výběrem jednoho náhodného obrázku od každého produktu dokázali zlepšit úspěšnosti modelů. Při náhodném výběru však stále dochází k výběru nevhodných obrázků, pouze se omezí množství nevhodných obrázků předkládaných síti. Lepším řešením by však bylo pokusit se vybírat ty dobré – reprezentativní – obrázky a síť trénovat na nich. Bohužel však nejsme schopní implementovat jednoznačný výběr nejlepšího obrázku, protože nemáme nástroj, který by toto dokázal zhodnotit. Výběru nejlepšího obrázku se dokážeme nejvíce přiblížit použitím našeho trénovaného modelu na obrázky u produktu a vybrat z nich ten, který nám vrátí největší pravděpodobnost správné kategorie. Tedy obrázek, s jehož pomocí dokážeme produkt nejúspěšněji správně zařadit.

Stejně jako v předchozí části je výběr obrázku pro trénování implementován ve funkci, která načítá data ze souboru. U každého produktu nejdříve načteme všechny obrázky, převedeme je to potřebného formátu, a následně na ně použijeme funkci *predict*. Ta nám pro každý obrázek vrátí pravděpodobnosti jednotlivých kategorií. Poté porovnáme hodnoty



Obrázek 6.5: Celkové úspěšnosti a chyby jednotlivých sítí v průběhu trénování. Při trénování byly všechny obrázky daného produktu sloučeny do jednoho.

pravděpodobností kategorie, do které produkt opravdu patří, a obrázek, který má tuto hodnotu nejvyšší, pak předáme síti k trénování.

Při trénování byla použita optimalizační metoda Adam s hodnotou learning rate 0,0001. V prvních dvou epochách byly trénovány pouze poslední vrstvy sítí. Od třetí etapy pak probíhalo trénování celých sítí. Bohužel kvůli použití predikce pravděpodobností vedle vlastního trénování nám narostla časová náročnost učení. U modelu Inception v3 nám jedna trénovací epocha zabere asi 106 hodin, u modelu MobileNet trvá jedna epocha přes 43 hodin.

Pokud jde o model Inception-ResNet v2, jeho trénování je stejně jako v předchozích částech výrazně časově náročnější než zbylé dva modely, konkrétně je to téměř 175 hodin za první epochu trénování. V této epoše navíc ani nebyl trénován celý model, pouze jeho poslední vrstvy. Po dokončení této epochy bylo tedy trénování tohoto modelu zastaveno.

Jak je vidět z grafu na obrázku 6.4, přestože modely Inception v3 i MobileNet se učí dobře a ukazují trochu větší potenciál růstu úspěšnosti než modely trénované s náhodným výběrem obrázků, není zde takový rozdíl, jaký bychom mohli čekat. To je pravděpodobně způsobeno tím, že obrázky pro trénování vybíráme podle úspěšnosti klasifikace již od samého začátku. Pokud máme model, který má na začátku nulovou úspěšnost, budou pravděpodobnosti správné kategorie u jednotlivých obrázků podobné a blízké nule. Výběr obrázku s nejvyšší pravděpodobností tedy může v některých případech probíhat spíše náhodně než aby reálně reflektoval vhodnost jednotlivých obrázků. Řešením by bylo nejdříve natrénovat klasifikátor na rozumnou úspěšnost, například právě pomocí výběru náhodného obrázku, a teprve poté použít trénování s výběrem nejúspěšnějšího obrázku.

6.5 Sloučení obrázků do jednoho

V předchozích částech jsme problém s více obrázky pro jeden produkt řešili omezením trénování pouze na jeden obrázek od každého produktu. To nám sice vyloučí obrázky nevhodně reprezentující produkt z procesu trénování, aby nám ho negativně neovlivňovaly, ale zároveň přitom samozřejmě také ztrácíme část informací o produktu. Alternativním řešením je poskytnout všechny obrázky síti naráz, aby si vytvořila kompletní přehled o produktu. Tímto způsobem nejen, že bude mít síť vždy k dispozici reprezentativní obrázky produktu, ale i případné doplňující informace o něm.

Jedním způsobem, jak tohoto dosáhnout, je sloučit všechny obrázky patřící ke stejnému produktu do jednoho. Stejně jako výběr v předchozích případech bylo toto slučování

implementováno hned při načítání dat ze souboru. Pro každý produkt se před začátkem načítání jednotlivých obrázků vytvoří prázdný (nulový) obrázek s dvojnásobnými rozměry (tedy pokud původní velikost je 180x180, tento obrázek bude mít velikost 360x360). V něm si můžeme přestavit čtyři pozice pro načítané obrázky. Obrázky na tyto pozice pak vložíme náhodně, přičemž, pokud produkt obsahuje méně než čtyři obrázky, zbylé pozice zůstávají nulové. Při trénování modelu sítě Inception v3 jsou takto vytvořené obrázky předané modelu k trénování, při trénování modelu MobileNet jsou ještě nejdříve zmenšeny na velikost 224x224. Síť Inception-ResNet v2 už vzhledem k neúspěchům v předchozích experimentech nebyla použita. Takto sloučené obrázky byly pak použity i při testování.

Při trénování sítě Inception v3 byl learning rate na začátku nastaven na hodnotu 0,001. Po skončení druhé epochy byl pak snížen na hodnotu 0,0005. Již od začátku byl přitom trénován celý model zaráz, neboť použitím sloučených obrázků se již odchylujeme od podoby dat, na nichž byla síť před-trénována a předpokládáme tedy, že ji bude potřeba částečně přeučit. Jak je vidět z grafu na obrázku 6.5 síť se bez problémů začala velmi rychle učit. Problém byl však znovu s časovou náročností učení, jedna tréninková epocha sítě trvá dokonce více než 155 hodin.

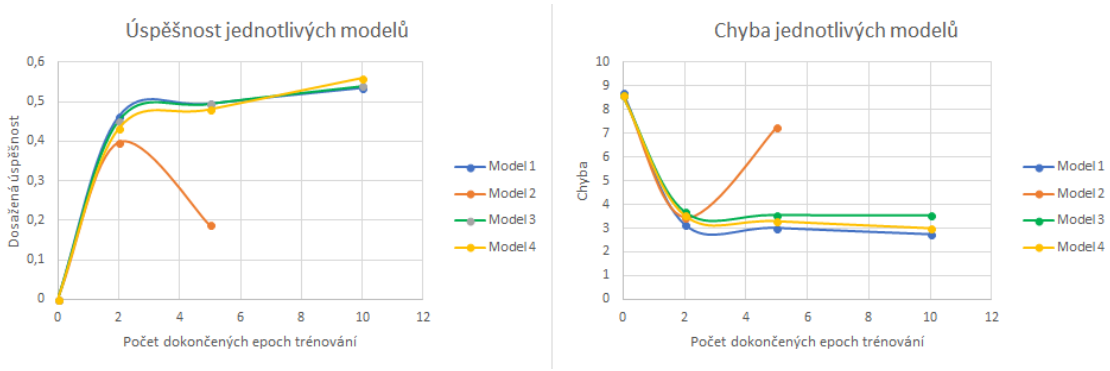
Začátek trénování sítě MobileNet probíhal stejně. I tato síť byla od začátku trénována celá a počáteční hodnota learning rate byla 0,001. Snížení learning rate na hodnotu 0,0005 zde však proběhlo až po skončení třetí epochy, po skončení desáté epochy pak byl pak snížen na hodnotu 0,00001. U tohoto modelu probíhalo učení rychle, jedna epocha trvala pouze 33 hodin. Tento rozdíl oproti modelu Inception v3 je pravděpodobně z velké části způsobem menší velikostí vstupního obrázku. Jak vidíme na obrázku 6.5, síť se sice učí relativně pomalu, ale stabilně a dá se předpokládat, že kdyby se ji podařilo dotrénovat dosáhla by dobrých výsledků.

Jak jsme tedy zjistili, sloučením všech obrázků produktu do jednoho, jsme schopni modely trénovat velmi dobře. Dalo by se i očekávat, že takto natrénovaný model dosáhne vyšší úspěšnosti než modely pracující pouze s jedním z obrázků. Modely se však učí pomaleji, na úspěšné dotrénování bychom potřebovali podstatně více epoch, a i časová náročnost učení je zde vyšší.

6.6 Sloučení obrázků v kombinaci s rozdělením produktů podle počtu obrázků

Jak se ukázalo v předchozí části, sloučení obrázků sice vede k úspěšnému trénování sítí, toto trénování je ale velmi časově náročné. Čas potřebný k natrénování sítě by se však snížil, kdyby některé části trénování mohly probíhat paralelně. Toho by se dalo dosáhnout například rozdělením datasetu na několik částí a na každé části natrénovat zvláštní model. Trénování těchto jednotlivých částečných modelů pak může být puštěno zaráz.

Při takovémto rozdělené dat se nabízí rozdělit produkty podle počtu obrázků. Z trénovacího datasetu nám tedy vzniknou čtyři menší, přičemž první obsahuje produkty s pouze jedním obrázkem, druhý produkty se dvěma obrázky, třetí produkty se třemi obrázky a čtvrtý produkty se čtyřmi obrázky. Na nich pak natrénujeme čtyři modely, které zde máme pojmenované jako Modely 1-4 podle počtu obrázků u produktů, na nichž se trénují. Pro snadné vyhodnocení úspěšnosti těchto modelů byl stejně jako trénovací dataset rozdělen i validační dataset a modely byly vyhodnocovány vždy jen na části validačních dat s odpovídajícím počtem obrázků u produktů.



Obrázek 6.6: Celkové úspěšnosti a chyby jednotlivých sítí v průběhu trénování. Každý z modelů byl trénován na jiné části dat. Při trénování byly všechny obrázky daného produktu sloučeny do jednoho.

Kombinování obrázků pak probíhalo tak, že u produktů s jedním obrázkem byl prostě použit tento obrázek, u produktů se dvěma a třemi obrázky byly obrázky spojeny vedle sebe do řady a u produktů se čtyřmi obrázky byly obrázky složeny do matice 2x2. Z tohoto důvodu byl v tomto experimentu použit pouze model Inception v3.

Při trénování byl pak použit nejdříve learning rate 0,001, který byl po dokončení druhé epochy snížen na 0,0005 a po dokončení sedmé epochy následně snížen na 0,00001. U všech modelů se od začátku trénovala celá síť naráz. Časová náročnost výpočtu se nám přitom opravdu snížila, u Modelu 1 trvá jedna epocha zhruba 24,5 hodiny, u Modelu 2 asi 12 hodin, u Modelu 3 něco přes 8,5 hodiny a u Modelu 4 něco přes 22,5 hodin. Čas potřebný pro jednu epochu tedy není u jednotlivých modelů příliš vyrovnaný, oproti 155 hodinám, které síť Inception v3 potřebovala na jednu epochu v předchozím experimentu, se však jedná o velmi výrazné zrychlení.

Když se podíváme na úspěšnost jednotlivých modelů v průběhu trénování (obrázek6.6), zjistíme, že Model 1, Model 3 a Model 4 mají téměř stejný vývoj a poměrně rozumně, i když pomalu, se učí. Úspěšnost Modelu 2 však po dokončení druhé epochy začne prudce klesat. Bohužel se nepodařilo zjistit, čím je to způsobené. Nezdá se, že by to bylo v důsledku postupu učení, protože ten funguje přibližně stejně pro všechny ostatní modely. Nedochází ani k přetrénování sítě a nepodařilo se odhalit žádný zjevný problém s trénovacími ani validačními daty tohoto modelu. Z tohoto důvodu tedy musíme tento experiment prohlásit z neznámého důvodu za neúspěšný.

Kapitola 7

Závěr

Cílem této práce bylo pokusit se pomocí hlubokých neuronových sítí vytvořit co nejúspěšnější klasifikátor reálných fotografií. Přitom byla využita data za soutěže *Cdiscount's Image Classification Challenge*.

Ze všeho nejdříve bylo potřeba provést analýzu těchto dat. Přitom bylo zjištěno, že data obsahují množství problémů, které znesnadňují učení sítí. Dále jsme potom datovou sadu rozdělili na dvě části – trénovací a validační, abychom mohli data využít při trénování sítí. Následně jsme ještě museli navrhnout a implementovat způsob načítání dat do operační paměti, neboť datová sada je příliš velká na to, aby se s ní dalo pracovat vcelku.

V další části pak bylo potřeba vytvořit modely, na kterých budeme provádět experimenty. Přitom byly použité před-trénované hluboké konvoluční sítě *Inception v3*, *Inception-ResNet v2* a *MobileNet*, vytvořené společností Google. Modely jsme vytvořili pomocí konstruktorů zahrnutých v knihovně Keras. Následně jsme je upravili, aby odpovídali našim konkrétním požadavkům.

V poslední části práce jsme navrhli, provedli a zhodnotili experimenty. Cílem experimentů bylo najít funkční postupy trénování našich zvolených modelů na našem datasetu. Přitom jsme využívali především zjištěných informací o podobě datasetu, neboť jak jsme zjistili hned na začátku, bez aplikování těchto informací nevedlo trénování modelů k žádnému růstu jejich úspěšnosti při klasifikaci.

Některé z experimentů vedly ke znatelnému zlepšení při trénování modelů, přesto se nám však nepodařilo dosáhnout úspěšnosti, ve kterou jsme doufali. To bylo způsobeno především tím, že trénování modelů na našich datech se ukázalo jako příliš časově náročné (v Metacentru bylo v rámci práce spotřebováno přes 446 dnů procesorového času) a v rámci této práce jsme nestihli modely dotrénovat až do konce. Odhadem by bylo potřeba modely trénovat 2-3krát déle, než jsme měli možnost.

Na základě provedených experimentů bych se při práci s takovouto datovou sadou, kde jednotlivé položky obsahují více než jeden obrázek, přiklonila spíše k metodě slučování obrázků do jednoho a trénování modelů s využitím kompletních informací o položkách. Trénování modelů by mohlo být dále rozšířeno o úpravy vstupních obrázků, jako například výřezy nebo překlopení obrázku.

Literatura

[1] Arora, S.; Bhaskara, A.; Ge, R.; aj.: Provable Bounds for Learning Some Deep Representations. 2013, arXiv:1310.6343.

[2] Bhandare, A.; Bhide, M.; Gokhale, P.; aj.: Applications of Convolutional Neural Networks. *International Journal of Computer Science and Information Technologies*, 2016: s. 2206–2215, ISSN 0975-9646.

[3] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] Gwardys, G.: Convolutional Neural Networks backpropagation: from intuition to derivation. [Online; navštíveno 29.04.2018].
URL https://grzegorzgwardys.wordpress.com/2016/04/22/8/

[5] Havránek, V.: *Umělé neuronové sítě a zpětnovazební učení*. Diplomová práce, Univezita Karlova v Praze, Matematicko-fyzikální fakulta, Praha, 2008.

[6] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. 2015, arXiv:1512.03385.

[7] Howard, A. G.; Zhu, M.; Chen, B.; aj.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017, arXiv:1704.04861.

[8] Hubel, D.; Wiesel, T.: Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 1962: s. 106–154.

[9] Jaderberg, M.; Vedaldi, A.; Zisserman, A.: Speeding up Convolutional Neural Networks with Low Rank Expansions. 2014, arXiv:1405.3866.

[10] Jin, J.; Dundar, A.; Culurciello, E.: Flattened Convolutional Neural Networks for Feedforward Acceleration. 2014, arXiv:1412.5474.

[11] Kafunah, J.: Backpropagation In Convolutional Neural Networks. [Online; navštíveno 29.04.2018].
URL http://jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

[12] Karpathy, A.: Convolutional Neural Networks. [Online; navštíveno 24.4.2018].
URL http://cs231n.github.io/convolutional-networks/

[13] Krajčovičová, M.: *Konvoluční neuronová síť pro zpracování obrazu*. Diplomová práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Brno, 2015.

[14] Lin, M.; Chen, Q.; Yan, S.: Network In Network. 2013, arXiv:1312.4400.

[15] Nielsen, M. A.: Neural Networks and Deep Learning. 2015, [Online; navštíveno 24.4.2018].
URL http://neuralnetworksanddeeplearning.com/

[16] Ruder, S.: An overview of gradient descent optimization algorithms. 2016, arXiv:1609.04747.

[17] Rumelhart, D.; Hinton, G.; Williams, R.: Learning representations by back-propagating errors. *Nature*, 1986: s. 533–536.

[18] Srivastava, N.; Hinton, G.; Krizhevsky, A.; aj.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014: s. 1929–1958, ISSN 1533-7928.

[19] Szegedy, C.; Ioffe, S.; Vanhoucke, V.: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. 2016, arXiv:1602.07261.

[20] Szegedy, C.; Liu, W.; Jia, Y.; aj.: Going Deeper with Convolutions. 2014, arXiv:1409.4842.

[21] Szegedy, C.; Vanhoucke, V.; Ioffe, S.; aj.: Rethinking the Inception Architecture for Computer Vision. 2015, arXiv:1512.00567.

[22] Wu, J.; Leng, C.; Wang, Y.; aj.: Quantized Convolutional Neural Networks for Mobile Devices. 2015, arXiv:1512.06473.

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