



ZADÁNÍ BAKALÁ SKÉ PRÁCE

Název:	Detekce objektů v ptáčích hnízdech pomocí neuronových sítí
Student:	Jan Havlík
Vedoucí:	Ing. Josef Pavlásek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Navhněte a implementujte knihovnu umožňující zpracovávat data z kamery umístěné v hnízdě (ptačí budce) s cílem rozpoznat počet vajec v hnízdě. Pro rozpoznání použijte existující algoritmy umělé inteligence využívající data získaná ze serveru PtaciOnline.cz a další sv. nástroje projektu BirdObserver (athena.pef.czu.cz).

Postupujte v těchto krocích:

1. Provejte detailní specifikaci požadavku.
2. Seznámte se s projektem BirdObserver a strukturou dat na serveru PtaciOnline.cz.
3. Provejte analýzu a návrh knihovny.
4. Návrh implementujte, zdokumentujte a vhodným způsobem otestujte.
5. Knihovnu navrhněte a implementujte v jazyce Java tak, aby bylo možné ji formou závislostí (dependency) integrovat s ostatními nástroji projektu BirdObserver.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrďák, CSc.
d. kan

V Praze dne 12. ledna 2017



**FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE**

Bakalářská práce

Detekce objektů v ptačích hnízdech pomocí neuronových sítí

Jan Havlůj

Katedra softwarového inženýrství
Vedoucí práce: Ing. Josef Pavláček, Ph.D.

2. ledna 2018

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce, panu Ing. Josefovi Pavláčkovi Ph.D. za jeho čas, ochotu a pomoc při vedení této práce. Děkuji svým rodičům a přítelkyni, kteří mi byli po celou dobu studia velkou oporou. V neposlední řadě bych chtěl poděkovat svým spolužákům, kteří mě při studiu nikdy neváhali podpořit.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 2. ledna 2018

.....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2018 Jan Havlůj. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

0.0.1 Odkaz na tuto práci

Havlůj, Jan. *Detekce objektů v ptačích hnizdech pomocí neuronových sítí*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018. Dostupný také z WWW: <https://github.com/havlj/bachelor_thesis>.

Abstrakt

Tato práce se zabývá návrhem a tvorbou softwarové knihovny pro detekci objektů v obrazu použitím neuronových sítí.

V první části jsme seznámeni s cílem práce a detailní specifikací požadavků. Druhá část se věnuje teorii počítačového vidění, strojového učení a neuronových sítí. Následně je na základě získaných teoretických znalostí a specifikace požadavků vypracována analýza možného řešení. Dle analýzy jsou navržena a implementována dvě řešení. První implementace se zaměřuje na detekci objektů, druhá na rozpoznávání a klasifikaci obrazu. V poslední části je vybráno efektivnější řešení, které je řádně ověřeno a otestováno.

Výsledkem je softwarová knihovna, která umožňuje automaticky rozpoznávat počet vajec v daném videu. Celý program je implementován v jazyce Java a je možné ho integrovat do projektu BirdObserver.

Klíčová slova neuronové sítě, počítačové vidění, detekce objektů, strojové učení, učení s učitelem, rozpoznávání obrazu, detekce vajec, ptačí hnízda, Java

Abstract

This thesis focuses on designing and creating a software library that is able to detect objects in an image by using neural networks.

In the first part of the thesis, goals are set technical parameters are specified. The second part discusses the theory behind computer vision, machine learning, and neural networks. Using the technical specification and acquired theoretical knowledge, a detailed analysis of a possible solution is presented. There are two implementations of the analysis. The first one is using object detection to meet it's goal, the second one image recognition. In the last part of the thesis, a more effective implementation is chosen, which is then properly tested and verified.

The result of this thesis is a software library that is able to automatically detect the number of eggs in a given video sequence. The entire solution is written in Java and is easily intergratable with the BirdObserver project.

Keywords neural networks, computer vision, object detection, machine learning, supervised learning, image recognition, egg detection, bird nests, Java

Obsah

Úvod	15
Struktura práce	15
1 Cíl práce	17
1.1 Nefunkční požadavky	17
1.2 Funkční požadavky	17
2 Rešerše	19
2.1 Počítačové vidění	19
2.2 Segmentace obrazu	19
2.3 Detekce objektů	19
2.4 Klasifikace obrazu	19
2.5 Neuronové sítě	19
2.6 Tensorflow	19
3 Analýza a návrh	21
3.1 Ptáci Online	21
3.2 Představení vstupních dat	21
3.3 Současný způsob zpracování dat	22
3.4 Nefunkční požadavky	23
3.5 Funkční požadavky	23
3.6 Návrh řešení	25
3.7 Volba technologií	26
3.8 Shrnutí kapitoly	27
4 Nástroje	29
4.1 Hromadné stažení dat	29
4.2 Příprava trénovacích a testovacích dat	30
5 Implementace detekování objektů	35
5.1 Příprava vývojového prostředí	35
5.2 Příprava dat	36
5.3 Struktura projektu	36

5.4	Trénování neuronové sítě	36
5.5	Ověření funkčnosti a výsledky	36
5.6	Závěr	37
6	Implementace rozpoznávání obrazu	39
6.1	Příprava dat	39
6.2	Trénování neuronové sítě	41
6.3	Ověření funkčnosti a výsledky	42
6.4	Závěr	42
7	Implementace Java knihovny	43
7.1	Srovnání implementací neuronových sítí a volba řešení	43
7.2	Použití exportovaného modelu v Javě	43
7.3	Uživatelské rozhraní knihovny	43
7.4	Implementace??	43
7.5	Maven	43
7.6	Výsledek	43
8	Ověření implementace	45
Závěr		47
Literatura		49
A	Seznam použitých zkratek	51
B	Dokumentace API knihovny	53
B.1	Package org.cvut.havluja1.eggdetector	53
C	Tagger	59
C.1	Dokumentace (v AJ)	59
C.2	Zdrojový kód	59
C.3	Ukázkový výstup	59
D	Folder Trimmer	61
D.1	Dokumentace (v AJ)	61
D.2	Zdrojový kód	61
E	Přetrénování finální vrstvy modelu Inception-v4	65
F	Použité programy	77
G	Obsah přiloženého CD	79

Seznam obrázků

3.1	Ukázka neupraveného snímku z ptačí budky.	22
3.2	Vejce nemusí být vždy viditelná.	22
3.3	Program určený ke klasifikaci obrazu.	25
4.1	Tagger – hromadné určování počtu vajec.	31
4.2	Uživatelské rozhraní nástroje LabelImg.	33

Seznam algoritmů

3.1	Prvotní návrh uživatelského rozhraní knihovny.	24
4.1	Hromadné stažení dat ze serveru athena.pef.czu.cz.	29
4.2	Výběr vhodných složek pro trénování nástrojem Tagger.	32
	media/labelimg.xml	33
	media/imgdata.xml	59
D.1	Zdrojový kód nástroje FolderTrimmer	62
E.1	Přetrénování finální vrstvy modelu Inception-v4.	65

Úvod

Projekt Ptáci Online byl spuštěn Fakultou životního prostředí České zemědělské univerzity v Praze roku 2014 [1]. Hlavním cílem projektu je poskytnout vědecká data, ve formě videa z ptačích budek, široké veřejnosti [1]. Projekt se těší poměrně velké popularitě [2] a aktuálně spolupracuje s více než dvěma desítkami spolupracovníků. Mezi ně patří lidé z akademické sféry, soukromého sektoru, ale i z Ministerstva životního prostředí České republiky [1]. Vzhledem k množství pořízených záznamů, by bylo žádoucí informace extrahovat automaticky pomocí algoritmů umělé inteligence. Předmětem této práce je vytvořit algoritmus, který je schopný určit počet vajec v hnizdě v daném videozáznamu.

Existuje hned několik způsobů, jak naučit počítač „vidět“. Téměř vždy se musíme zaměřit na předzpracování obrazu, jeho vlastnosti a jeho segmentaci. Jako řešení pak můžeme zvolit různou sadu deterministických algoritmů, například pro detekci hran nebo na samotné klasifikování segmentovaného obrazu. Všechny tyto algoritmy mají však jednu společnou nevýhodu. Formálně popsat tvar nějakého objektu a vytvořit sadu pravidel, podle kterých poznáme, jestli se jedná o hledaný objekt, je velmi těžké. Světelné podmínky nemusí být ideální, objekty se mohou překrývat, mohou být různě barevné, vzdálené nebo otočené. Všechny tyto problémy znemožňují vytvoření perfektního algoritmu manuálně. Lepším řešením je manuálně vytvořit jeden algoritmus, který dokáže „vytrénovat“ druhý obecný algoritmus k vyřešení konkrétního problému. Za tímto účelem vznikly algoritmy inspirované přírodou, například umělé neuronové sítě inspirované lidským učením a mozkem nebo evoluční algoritmy inspirované evoluční teorií. V této práci se budeme soustředit na použití neuronových sítí pro vyřešení problémů počítačového vidění.

Struktura práce

Práce je rozdělena do 7 částí. První část obsahuje stanovení cílů a specifikaci funkčních i nefunkčních požadavků. Druhá část je teoretická. Diskutuje problematiku počítačového vidění, strojového učení a neuronových sítí. Třetí, analytická část, se zaměřuje na výběr vhodných technologií a postupů pro tvorbu implementace. Čtvrtou i pátou, praktickou část, tvoří dva různé způsoby implementace, jejich porovnání, otestování a validace. Na závěr vybereme efektivnější implementaci, dle které je vytvořena softwarová knihovna v jazyce Java.

Cíl práce

Cílem teoretické části práce je se seznámit s technologiemi a principy, které jsou dnes standardně používány k řešení problémů spjatých s počítačovým viděním a strojovým učením. Zaměříme se především na umělé neuronové sítě, jejich historii, současný vývoj a hlavně na principy jejich fungování. Na základě získaných teoretických znalostí je vypracována analýza řešení, která může být považována za výstup teoretické části. Cílem analýzy je jednak selekce vhodných principů a technologií pro implementaci řešení, ale i zpracování technických požadavků zároveň s popsáním současného stavu.

Cílem praktické části je implementovat řešení, které bude dostatečně rychlé a spolehlivé. Výstupem bude softwarová knihovna, která bude umožňovat rozpoznávat počet vajec v hnázde na jednotlivých snímcích i na sekvenčních videa. Součástí knihovny bude neuronová síť, která bude řešit samotnou detekci. Tento přístup umožní recyklaci výsledného programu s minimální modifikací pro řešení podobných problémů, jako například určení druhu ptáka, počtu mláďat nebo predaci hnázda.

Následující seznam funkčních a nefunkčních požadavků slouží k přesné definici požadavků na výsledný systém. Tento seznam byl vytvořen ve spolupráci s vedoucím práce.

1.1 Nefunkční požadavky

N1 Knihovna musí pracovat v reálném čase. Po zadání vstupních dat je výsledek očekáván maximálně v jednotkách sekund.

1.2 Funkční požadavky

F1 Systém bude distribuovaný formou Java knihovny v archivu JAR¹.

F2 Knihovna bude distribuována ve dvou verzích:

- JAR archiv obsahující jak přeložený zdrojový kód knihovny, tak i všechny závislosti², které jsou knihovnou vyžadovány.

¹Zkratka pro Java ARchive. „JAR je kompresní souborový formát, používaný platformou Java, založený na ZIP kompresi.“[3]

²Zdrojový kód může používat funkce poskytované jinými softwarovými knihovnami.

1. CÍL PRÁCE

- JAR archiv obsahující pouze přeložený zdrojový kód knihovny bez externích závislostí. Do archivu bude přibalen konfigurační soubor pro systém Maven[4] s definicí závislostí.

F3 Knihovna musí umět pracovat se strukturou dat na serveru <http://athena.pef.czu.cz/ptacionline/>.

F4 Knihovna musí umět určit počet vajec v hnízdě pro každy jednotlivý snímek.

F5 Knihovna musí umět určit počet vajec v hnízdě pro složku jako celek. Složka se skládá ze sekvence obázků, které reprezentují jednu videosekvenci.

F6 Snímky mohou být ve formátu JPEG³ a PNG⁴.

³Joint Photographic Experts Group.

⁴Portable Network Graphics.

Rešerše

2.1 Počítačové vidění

2.2 Segmentace obrazu

2.3 Detekce objektů

2.4 Klasifikace obrazu

jak funguje

2.5 Neuronové sítě

2.5.1 CNN

2.5.2 RCNN

2.6 Tensorflow

Text

Analýza a návrh

V této kapitole se zaměříme na strukturu projektu Ptáci Online, problémy spojené se současným způsobem sbírání dat, rozbor funkčních a nefunkčních požadavků, návrh řešení a výběr technologií. Na závěr budeme diskutovat dva různé způsoby řešení – jejich výhody a nevýhody.

3.1 Ptáci Online

„Cílem projektu je popularizovat ochranu ptáků v blízkosti lidských sídel, jejich hnízdění, včetně jeho monitoringu, s využitím speciálního technického zařízení tzv. „chytré ptačí budky“.“ [5]

Tyto „chytré ptačí budky“ slouží k monitorování hnízdícího ptactva. Každá budka obsahuje jednu nebo dvě kamery s nočním přísvitem. Ve vletové otvoru budky je umístěn pohybový senzor, který spustí natáčení kamér při detekci pohybu. Dále je do budky vestavěn venkovní a vnitřní teplotní senzor, mikrofon a senzor venkovního osvětlení, který reguluje funkci přísvitu kamér. Přenos nasbíraných dat z budky probíhá přes ethernetový PoE⁵ kabel. Tento kabel zajišťuje i napájení veškeré elektroniky uvnitř budky. [1] Jak vypadá záznam z budky je vidět na obr. 3.1.

3.2 Představení vstupních data

Ptačí budka vyprodukuje sekvenci videa pokaždé, když je v ní detekován pohyb. Kvůli energetické úspornosti kamer a množství přenášených dat mají tyto videa nižší snímkovou frekvenci⁶. Z takového videozáznamu lze extrahovat množinu jednotlivých snímků, které jako celek tvoří dané video. Snímky jsou ukládány do formátu PNG a zařazeny do složek, přičemž **jedna složka reprezentuje jeden videozáznam**.

Výsledná softwarová knihovna bude umět pracovat s jednotlivými složkami. Načte všechny snímky z dané složky a následně je zpracuje. Uživatel bude schopen získat informace o složce jako celku i jednotlivých snímcích.

⁵Power over Ethernet.

⁶Počet snímků za sekundu.

3. ANALÝZA A NÁVRH



Obrázek 3.1: Ukázka neupraveného snímku z ptačí budky.



(a) Vejce jsou zřetelně viditelná v čase 0:01.



(b) Vejce nejsou viditelná ve zbytku videa, jako je tomu např. v čase 0:14.

Obrázek 3.2: Vejce nemusí být vždy viditelná.

3.3 Současný způsob zpracování dat

Akademickí pracovníci potřebují nashromáždit velké množství dat, ale taková činnost je velmi časově náročná. Proto jsou na takovou práci najímáni brigádníci. Brigádníky je nutno nejprve zaškolit, aby věděli, co mají ve videích hledat. Poté sledují jedno video za druhým a získané informace zapisují do tabulek v Excelu. Tento způsob práce je drahý a časově náročný.

Jedna z informací, která nás může zajímat, je počet vajec v hnizdě. Brigádník musí video otevřít, shlédnout a najít část, kde jsou vejce zřetelně viditelná (viz obr. 3.2a). Poté vejce spočítá a výsledek zapíše do tabulky. Jelikož je nutné získat co nejvíce dat, brigádníci tento proces vykonávají co nejrychleji. To vede k chybám, např. chybně spočítaný počet vajec nebo zapsání výsledku na špatný řádek tabulky.

Je snahou vytvořit systémy, které by tyto úkoly mohly provádět automaticky. Příkladem takového systému je například práce od Pavla Šumy, který se snaží automaticky zjistit počet mláďat v hnizdě [6]. Dalším příkladem je i tato práce.

3.4 Nefunkční požadavky

3.4.1 N1

Knihovna musí pracovat v reálném čase. Po zadání vstupních dat je výsledek očekáván maximálně v jednotkách sekund.

Z vlastnosti neuronových sítí vyplývá, že tento požadavek nebude problém splnit. Samotný průchod snímku grafem, resp. průchod snímku neuronovou sítí není příliš časově náročný. Nutnou podmínkou však je dostupný soubor obsahující popis struktury neuronové sítě. Trénování, nebo-li hledání optimální struktury neuronové sítě je velmi výpočetně náročná činnost, která musí být dokončena **před** použitím knihovny.

3.5 Funkční požadavky

3.5.1 F1

Systém bude distribuovaný formou Java knihovny v archivu JAR.

Neuronové sítě můžeme naprogramovat v libovolném programovacím jazyce. Tato podmínka může být tedy bez problému splněna. Výsledná softwarová knihovna pro detekci počtu vajec v hnizdě bude implementována v jazyce Java.

3.5.2 F2

Knihovna bude distribuována ve dvou verzích:

- *JAR archiv obsahující jak přeložený zdrojový kód knihovny, tak i všechny závislosti, které jsou knihovnou vyžadovány.*
- *JAR archiv obsahující pouze přeložený zdrojový kód knihovny bez externích závislostí. Do archivu bude přibalen konfigurační soubor pro systém Maven[4] s definicí závislostí.*

Distribuci přeloženého zdrojového kódu vyřešíme pomocí nástroje pro správu, řízení a automatizaci buildů aplikací. Maven je pro nás nejhodnější volba, vzhledem k druhé části podmínky – Do archivu bude přibalen konfigurační soubor pro systém Maven s definicí závislostí.

3.5.3 F3

Knihovna musí umět pracovat se strukturou dat na serveru <http://athena.pef.czu.cz/ptacionline/>.

Knihovna bude schopna pracovat se strukturou dat popsanou v kapitole 3.2. Uživateli bude schopen interagovat s knihovnou, která bude reflektovat strukturu dat projektu:

3. ANALÝZA A NÁVRH

Algoritmus 3.1: Prvotní návrh uživatelského rozhraní knihovny.

```
1 // Inicializujeme EggDetector. Knihovna se připraví pro
2 // zpracování sekvencí.
3 EggDetector eggDetector = new EggDetector();
4
5 // EggDetectoru předáme absolutní cestu k složce, kterou
6 // chceme vyhodnotit. EggDetector nám vrátí třídu
7 // SequenceClassifier, která obsahuje veškeré informace
8 // k dané složce.
9 SequenceClassifier sequenceClassifier = eggDetector.evaluate(new File("image_dir"));
10
11 // Zjistíme finální počet vajec v hnízdě pro danou složku.
12 System.out.println("final count: " + sequenceClassifier.getFinalCount());
13
14 // Můžeme zjistit počet vajec v jednotlivých snímcích.
15 System.out.println("individual scores: " + sequenceClassifier.getIndividualCounts());
16
17 // Ukončíme EggDetector - uvolníme informace o neuronové
18 // sítě z paměti. Instance EggDetectoru se stane nepoužitelná.
19 eggDetector.closeSession();
```

3.5.4 F4

Knihovna musí umět určit počet vajec v hnízdě pro každý jednotlivý snímek.

Výsledná knihovna bude uživateli umět poskytnout informace o jednotlivých snímcích – viz algoritmus 3.1.

3.5.5 F5

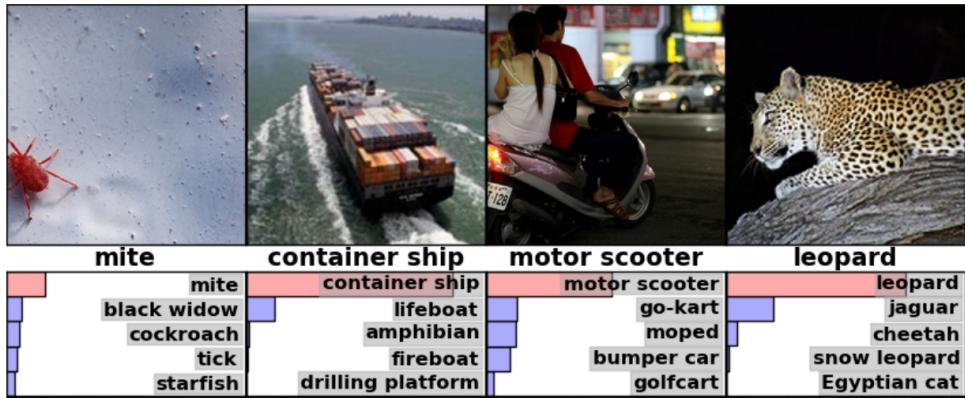
Knihovna musí umět určit počet vajec v hnízdě pro složku jako celek. Složka se skládá ze sekvence obázků, které reprezentují jednu videosekvenci.

Výsledná knihovna bude uživateli umět poskytnout informace o složce jako celku – viz algoritmus 3.1.

3.5.6 F6

Snímky mohou být ve formátu JPEG a PNG.

Pro zpracování snímků použijeme standartně dostupné třídy pro práci s grafikou v programovacím jazyce Java. Konkrétně `BufferedImage` a `Graphics2D` nám umožní zpracovat oba dva formáty – JPEG i PNG.



Obrázek 3.3: Program určený ke klasifikaci obrazu.

Zdroj: dokumentace softwarové knihovny TensorFlow [7].

3.6 Návrh řešení

Jádro implementace bude tvořit neuronová síť, která bude vytvořena metodou „učení s učitelem“⁷. Tvorba implementace se bude skládat ze čtyř částí:

- Příprava trénovacích a testovacích dat.
- Trénování neuronové sítě.
- „Konzumace“ vytrénované neuronové sítě knihovnou, která je výsledkem této práce.
- Ověření funkčnosti.

Existuje několik způsobů, jak neuronovou síť vytrénovat. Prvním možným řešením je neuronová síť určená ke klasifikaci celého obrazu. Typickým vstupem je snímek, ve kterém je objekt, který chceme rozpoznat, nejvýraznější. Pokročilejší typ této sítě je schopný popsat komplexnější scény, jako například „Dva lidé na pláži.“. Ukázka takového programu je vidět na obrázku 3.3. V našem případě bychom klasifikovali počet vajec v hnizdě, kde by jednotlivé výsledky reprezentovali počet vajec na daném snímku. Tzn. výstupem takové neuronové sítě by byla jedna z předem daných kategorií, o které si s největší pravděpodobností myslíme, že popisuje, co je na daném snímku. Každá kategorie by reprezentovala jiný počet vajec v hnizdě, například kategorie 0, kategorie 1, atd.

Druhým možným řešením je komplexnější neuronová síť určená k detekci objektů. Obraz je nejprve segmentován na části, o kterých si síť myslí, že by mohly obsahovat nějaké objekty. Následně jsou tyto části klasifikovány a je rozhodnuto, zda-li se jedná o objekt a s jakou pravděpodobností. Výsledkem je potom množina detekcí, která je tvořena umístěním objektu, typem objektu a pravděpodobností mírou, která reprezentuje jak moc je si síť jistá, že se opravdu jedná o daný objekt. Typickým vstupem je libovolný snímek, jako je tomu vidět na obrázku ???. V našem případě bychom hledali detekci vajec s relativně vysokou pravděpodobností. Počet detekcí by reprezentoval počet vajec v hnizdě.

Ať už zvolíme jakýkoliv způsob implementace, pro zpracování obrazu předáme neuronové síti pole hodnot o velikosti 300 x 300 x 3 (snímek bude zmenšen na velikost 300 x 300 bodů

⁷Supervised learning.

3. ANALÝZA A NÁVRH

a zůstane barevný – každý bod obsahuje 3 barevné složky⁸), které reprezentuje náš snímek. Knihovna nám poté vrátí požadovaný výsledek⁹.

3.7 Volba technologií

V této kapitole vybereme konkrétní platformy, programovací jazyky a nástroje potřebné k implementaci praktické části této práce.

3.7.1 Platforma

Vzhledem k funkčním požadavkům je jednoznačnou volbou programovací jazyk Java. Volíme verzi Java 8 SE, která poskytuje všechny nástroje, které k doručení výsledku potřebujeme.

3.7.2 Neuronové sítě

Pro značné usnadnění tvorby neuronových sítí použijeme softwarovou knihovnu. Mezi tři nejznámější patří TensorFlow, OpenNN a FANN¹⁰. Z těchto tří knihoven pouze TensorFlow poskytuje Java API a už jen z tohoto důvodu je náš nejlepší kandidát.

TensorFlow má skvělou dokumentaci [8] s velkým množstvím kompletních návodů [9]. Poskytuje API k detekci objektů, kde je možné využít již předtrénované modely [10]. Pomocné skripty určené k trénování neuronové sítě, uložení její struktury a ověření funkčnosti nainplementujeme v jazyce Python 3, jelikož je primárním jazykem pro práci s knihovnou TensorFlow.

3.7.3 Práce s daty

K vytrénování naší neuronové sítě je potřeba velké množství trénovacích dat. Trénovacími daty jsou myšleny snímky, ke kterým dodáme informace, jako například počet a umístění jednotlivých vajíček. Abychom si získávání a označování dat usnadnili, je potřeba několik nástrojů, které jsou detailně popsány v kapitole 4.

- Pro hromadné stažení dat použijeme nástroje `bash` a `wget`. Hromadně stažená data budou obsahovat i obsah, který pro nás není užitečný. Proto použijeme nástroj, který všechna neužitečná data smaže. Více v kapitole 4.1.
- Trénovací a testovací data vytvoříme pomocí nástrojů `LabelImg` a `Tagger`. Detailní informace obsahuje kapitola 4.2.

Abychom byli schopni nově vytvořená trénovací data použít k trénování neuronové sítě, musíme je převést do standardního formátu. V našem případě musíme z binárních dat snímků a textových XML souborů vytvořit tzv. `TFRecord` [11]. Každý způsob implementace vyžaduje mírně odlišný formát trénovacích dat. V kapitolách 5 a 6, které se popisují konkrétní implementace, diskutujeme i přípravu trénovacích dat.

⁸RGB - red, green, blue. Každý bod obrázku obsahuje informaci o koncentraci červené, zelené a modré.

⁹V případě rozpoznávání obrazu, knihovna vrátí seznam pravděpodobností pro všechny známe typy. V případě detekce objektů, knihovna vrátí seznam, pozici a typ objektů.

¹⁰Fast Artificial Neural Network.

3.8 Shrnutí kapitoly

V této kapitole jsme prozkoumali strukturu projektu Ptáci Online, problémy současného způsobu sbírání dat. Adresovali jsme všechny funkční i nefunkční požadavky s ohledem na strukturu projektu a dat. Navrhli jsme 2 možná konkurenční řešení, u kterých není předem jasné, jaké z nich je efektivnější. Nezbývá nám tedy nic jiného, než implementovat obě dvě řešení a jejich výsledky porovnat. Zaměřili jsme se také na výběr vhodných principů a nástrojů potřebných k úspěšné implementaci. Vybrali jsme platformu knihovny, knihovnu pro práci s neuronovými sítěmi a velkou škálu nástrojů pro přípravu dat.

V dalších kapitolách se zaměříme na samotné použití nástrojů a jejich tvorbu. Ale především budeme diskutovat oba dva způsoby implementace, které nakonec porovnáme.

Nástroje

Přípravu a zpracování dat si můžeme usnadnit pomocí několika nástrojů. Zaměříme se na hromadné stažení dat ze serveru <http://athena.pef.czu.cz/ptacionline/> a jejich „pročístění“. Dále si představíme nástroje, ve kterých obohatíme stažené snímky o informace potřebné k trénování neuronové sítě.

4.1 Hromadné stažení dat

4.1.1 Získání dat

Abychom nemuseli stahovat snímky jeden po druhém manuálně, pomůžeme si napsáním jednoduchého skriptu, který stáhne všechny snímky za nás. K tomu nám postačí dva nástroje: `bash` a `wget`. Tento skript stáhne veškerá data, která jsou na serveru <http://athena.pef.czu.cz/ptacionline/> dostupná. Detailní dokumentace skriptu je k nalezení v příloze G).

Algoritmus 4.1: Hromadné stažení dat ze serveru athena.pef.czu.cz.

```
1 #!/bin/bash
2
3 DIRECTORY=data
4 URL=http://athena.pef.czu.cz/ptacionline/
5
6 wget -o log.txt -nv --show-progress -c -P "$DIRECTORY" -r -np -nH --cut-dirs=1 -R
     index.html "$URL"
```

Stručné vysvětlení příkazu `wget`, který používáme na poslední řádce skriptu 4.1:

- Všechny složky a podsložky dostupné na serveru budou staženy lokálně do složky `$DIRECTORY`.
- `-o log.txt` vytvoří záznam do souboru `log.txt`.
- `-nv` zobrazuje pouze chyby, ne varování.
- `-show-progress` ukáže progres stahování.
- `-c` – pokračuj ve stahování nedokončených souborů.

4. NÁSTROJE

- **-r** – rekurzivně stahuj podsložky.
- **-np** – nestahuj soubory v složkách výše, než **ptacionline**.
- **-nH** – nestahuj do složky, která se jmenuje stejně jako doména, ale přímo do **\$DIRECTORY**.
- **-cut-dirs=1** – ve složce **\$DIRECTORY** vynech první složku (**ptacionline**).
- **-R index.html** – nestahuj soubory **.html**.

4.1.2 Čištění dat

Skript, který jsme představili v kapitole 4.1.1, stáhne veškerá data z daného serveru. Mezi takovými daty jsou snímky, které jsou pro nás užitečné, ale zbytek stažených dat je pro nás zbytečný. Abychom se v datech mohli lépe orientovat a ušetřit místo na pevném disku, bylo by vhodné nepotřebná data smazat. Pro tento účel naprogramujeme jednoduchý nástroj v programovacím jazyce Java, který automaticky ponechá data potřebná a data nepotřebná smaže.

Zdrojový kód a dokumentace nástroje **FolderTrimmer** je k nalezení v příloze D. Výsledný program stačí spustit a složku, která má být promazána, mu předat jako argument: **run.sh /home/demo/eggs/data**.

FolderTrimmer funguje ve dvou režimech. První, základní režim, smaže všechny soubory jiného typu než PNG, TXT a XML. V případě, že složka neobsahuje další složku nebo alespoň jeden soubor typu PNG, TXT nebo XML, bude také smazána. Druhý režim funguje stejně jako první, ale smaže všechny složky, které neobsahují soubor **imgdata.xml**. To umožní vymazání všech dat, které nejsou relevantní pro trénování neuronové sítě. První režim spustíme příkazem **run.sh false "cesta_ke_slozce"**. Druhý režim spustíme příkazem **run.sh true "cesta_ke_slozce"**.

4.2 Příprava trénovacích a testovacích dat

Když máme všechna potřebná data stažená, je potřeba je připravit tak, abychom je mohli použít k trénování neuronové sítě. Příprava dat se liší podle typu implementace, který zvolíme. V případě trénování neuronové sítě k detekci objektů, chceme k jednotlivým snímkům přidat informaci **kde, jaké velikosti a kolik** vajec se v nich nachází. V případě trénování neuronové sítě pro rozpoznávání (klasifikaci) obrazu, je potřeba ke snímkům přidat informaci o **celkovém počtu vajec**.

4.2.1 Tagger

Nástroj **Tagger** slouží k manuálnímu označování snímků. Pracuje na úrovni složek, kde jednotlivé složky a snímky reprezentují jednu videosekvenci. Uživatel pak může program spustit a poměrně rychle označit, kolik vajec se na daných snímcích nachází. Tagger je webovou aplikací s jednoduchým uživatelským rozhraním. Uživatel je prezentován všemi snímkům dané složky a má možnost u každého snímků specifikovat počet vajec. Uživateli je složka vybrána automaticky. Po odeslání dat je uživatel vyzván k označení další složky. Tento proces se opakuje do té doby, než jsou označené všechny složky. Ukázka uživatelského rozhraní je vidět na obrázku 4.1.

Výstupem programu jsou soubory **imgdata.xml**, které obsahují informace o počtu vajec na jednotlivých snímcích. Tyto data jsou pak používána pro trénování neuronové sítě k rozpoznávání obrazu, kde počet vajec reprezentuje možné výstupy neuronové sítě. Takto označené snímky

4.2. Příprava trénovacích a testovacích dat

[Different folder](#)

Prefill the form:

		<input type="text" value="0"/> <input type="button" value="▼"/>
		<input type="text" value="2"/> <input type="button" value="▼"/>
		<input type="text" value="2"/> <input type="button" value="▼"/>
		<input type="text" value="8"/> <input type="button" value="▼"/>
		<input type="text" value="8"/> <input type="button" value="▼"/>

Obrázek 4.1: Tagger – hromadné určování počtu vajec.

4. NÁSTROJE

se dají použít i pro validaci jakéholidiv řešení – finální softwarové knihovně předáme složku se snímky, knihovna vyhodnotí výsledky a my je poté můžeme porovnat s výsledky, které jsme manuálně nasbírali. Příklad výstupu je přiložen v příloze C.3.

Program je napsán v programovacím jazyce Java. Detailní popis nástroje se nachází v příloze C. Jedná se o webovou aplikaci, která je postavená na technologii Spring Boot [12]. Výpis 4.2 ukazuje algoritmus pro selekci vhodných složek.

Algoritmus 4.2: Výběr vhodných složek pro trénování nástrojem Tagger.

```
1 public static ArrayList<String> scanFolder(String location) {
2     File locFile = new File(location);
3
4     if (!locFile.exists()) {
5         return new ArrayList<>();
6     }
7
8     // example folder name: 20160430_073822_526_D
9     final Pattern pattern = Pattern.compile("\\d{8}_\\d{6}_\\d{3}_D");
10    List arr = Arrays.asList(locFile.list((File file, String name) -> {
11        File workingDir = new File(file.getAbsolutePath() + File.separator + name);
12        // needs to be a dir in a correct format
13        if (!workingDir.isDirectory() && !pattern.matcher(name).matches()) {
14            return false;
15        }
16
17        // if already tagged (contains imgdata.xml file)
18        File imgDataFile = new File(workingDir, "imgdata.xml");
19        if (imgDataFile.exists()) {
20            return false;
21        }
22
23        // contains any pictures
24        if (workingDir.list((f, n) -> {
25            File workingFile = new File(f.getAbsolutePath() + File.separator + n);
26            return workingFile.isFile() && FilenameUtils.getExtension(n).equals("png");
27        }).length <= 0) {
28            return false;
29        }
30
31        return true;
32    }));
33
34    return new ArrayList<>(arr);
35 }
```

4.2.2 LabelImg [13]

Trénovací data pro detekci objektů vyžadují jiný formát než data pro rozpoznávání obrazu. Je potřeba na jednotlivých snímcích označit pozici jednotlivých vajec. Přesně za tímto účelem byl vytvořen nástroj LabelImg [13]. Uživatel si při spuštění programu zvolí složku, ve které má data připravená na označení. Uživatel má možnost zadat typy objektů, které chce na snímcích

4.2. Příprava trénovacích a testovacích dat



Obrázek 4.2: Uživatelské rozhraní nástroje LabelImg.

označovat. V našem případě se jedná pouze o jeden typ objektu – vejce¹¹. Poté je uživatel prezentován se snímkem, kde má možnost objekty manuálně ohraňovat (viz obrázek 4.2).

Níže je přiložen ukázkový výstup programu LabelImg.

```

1 <annotation>
2   <folder>eggs</folder>
3   <filename>snap0001-0000000648.png</filename>
4   <path>C:\Users\test\Desktop\eggs\snap0001-0000000648.png</path>
5   <source>
6     <database>Unknown</database>
7   </source>
8   <size>
9     <width>1280</width>
10    <height>720</height>
11    <depth>3</depth>
12  </size>
13  <segmented>0</segmented>
14  <object>
15    <name>egg</name>
16    <pose>Unspecified</pose>
17    <truncated>0</truncated>
18    <difficult>0</difficult>
19    <bndbox>
20      <xmin>422</xmin>
21      <ymin>323</ymin>
22      <xmax>500</xmax>
23      <ymax>386</ymax>
24    </bndbox>
```

¹¹Ve skutečnosti se náš objekt (label) jmenuje egg.

4. NÁSTROJE

```
25 </object>
26 <object>
27     <name>egg</name>
28     <pose>Unspecified</pose>
29     <truncated>0</truncated>
30     <difficult>0</difficult>
31     <bndbox>
32         <xmin>462</xmin>
33         <ymin>379</ymin>
34         <xmax>544</xmax>
35         <ymax>447</ymax>
36     </bndbox>
37 </object>
38 </annotation>
```

Implementace detekování objektů

V této kapitole se zaměříme na implementaci neuronové sítě, která bude sloužit k detekci objektů. V rámci této práce nás bude zajímat pouze jeden objekt: vejce.

Abychom mohli vytrénovat nový model, který je schopný detekce vajec, je potřeba, abychom snímky a data převedli do speciálního formátu. Pro použití a validaci modelu budeme používat skript, u kterého bude vstupem obrázek libovolné velikosti. Interně bude zmenšen na velikost 300 x 300 bodů. Snímek bude reprezentován pole hodnot o velikosti 300 x 300 x 3 (snímek velikosti 300 x 300 bodů a zůstane barevný – každý bod obsahuje 3 barevné složky. Jakmile provedeme detekci objektů, musíme objekty klasifikovat do předem stanovených kategorií. Kategorie, do které budeme chtít snímky rozřadit, bude pouze **1**:

Kategorie „egg“: Objekt vejce.

Celé řešení – příprava dat, trénování neuronové sítě, testování funkčnosti a měření přesnosti budeme implementovat v programovacím jazyce Python 3.

5.1 Příprava vývojového prostředí

Abychom mohli používat TensorFlow Object Detection API, je potřeba následně připravit vývojové prostředí:

1. Nainstalovat Python 3 a pip3.
2. Stáhnout repositoř s TensorFlow modely: `git clone https://github.com/tensorflow/models`.
3. Nainstalovat modely: `cd models/research && python3 setup.py`.
4. Přidat tyto řádky do `~/.bashrc` (path_to_models_directory nahradíme skutečným umístěním modelů):

```
1  export MODELS=path_to_models_directory
2  export PYTHONPATH=$MODELS:$MODELS/slim
3  export OBJ_DET=$MODELS/object_detection
```

5. Nainstalovat TensorFlow: `sudo pip3 install tensorflow-gpu` nebo `sudo pip3 install tensorflow`. Více informací viz [16].

5. IMPLEMENTACE DETEKOVÁNÍ OBJEKTŮ

6. Pro ověření instalace spustíme interaktivní Python 3:

```
1 import tensorflow as tf
2 hello = tf.constant('EggDetector!')
3 sess = tf.Session()
4 print(sess.run(hello))
```

Systém by měl odpovědět „EggDetector!“.

5.2 Příprava dat

Nástrojem LabelImg (viz kapitola 4.2.2) jsem označil **1800** vajec. Data je potřeba rozdělit na trénovací a testovací. Testovací data slouží k validaci a vyhodnocení úspěšnosti trénovaného modelu. Standardně se data dělí přibližně v poměru 9:1 ve prospěch trénovacích dat. Data jsem rozdělil následovně:

- **1701** výskytů vajec jako trénovací data,
- **189** výskytů vajec jako data validační.

5.3 Struktura projektu

Projekt, ve kterém budeme celou implementaci tvořit, bude mít následující strukturu:

```
object-detection-training.....pracovní složka
|   |
|   +-- training.....složka s konfigurací
|   |   |
|   |   +-- checkpoint_model.....data modelu, ze kterého budeme vycházet
|   |   |
|   |   +-- egg_label_map.pbtxt
|   |   |
|   |   +-- pipeline.config
|   |
|   +-- test_images .....testovací obrázky k validaci
|   |
|   +-- frozen_graph.....výsledný, exportovaný graf
|   |
|   +-- images.....všechna testovací a trénovací data
|   |   |
|   |   +-- test .....testovací data
|   |   |
|   |   +-- train.....trénovací data
|   |
|   +-- export_inference_graph.py
|   +-- generate_tfrecord.py
|   +-- xml_to_csv.py
```

Stejná struktura je dostupná u přiloženého zdrojového kódu v příloze G, konkrétně ve složce `src/object-detection-training`.

5.4 Trénování neuronové sítě

struktura slozky

konfiguration file

muzeme trenovat lokalne nebo na cloudu

lokalne:

na cloudu
sledování trenování

5.5 Ověření funkčnosti a výsledky

ipnb notebook
ukázka výsledku trenování podle jednoho modelu a trenování podle druhého modelu

5.6 Závěr

Implementace rozpoznání obrazu

V této kapitole se zaměříme na implementaci neuronové sítě, která bude mít za cíl rozpoznat, do které kategorie snímek patří. Jedná se tedy o klasifikaci do předem daných kategorií. Vstupem pro naše skripty bude obrázek libovolné velikosti. Interně bude zmenšen na velikost 300 x 300 bodů. Snímek bude reprezentován pole hodnot o velikosti 300 x 300 x 3 (snímek velikosti 300 x 300 bodů a zůstane barevný – každý bod obsahuje 3 barevné složky. Kategorií, do kterých budeme chtít snímky rozřadit, bude **11**:

Kategorie „0“: Pro snímky, kde je počet vajec roven **0**.

Kategorie „1“: Pro snímky, kde je počet vajec roven **1**.

Kategorie „2“: Pro snímky, kde je počet vajec roven **2**.

Kategorie „3“: Pro snímky, kde je počet vajec roven **3**.

Kategorie „4“: Pro snímky, kde je počet vajec roven **4**.

Kategorie „5“: Pro snímky, kde je počet vajec roven **5**.

Kategorie „6“: Pro snímky, kde je počet vajec roven **6**.

Kategorie „7“: Pro snímky, kde je počet vajec roven **7**.

Kategorie „8“: Pro snímky, kde je počet vajec roven **8**.

Kategorie „9“: Pro snímky, kde je počet vajec roven **9**.

Kategorie „10“: Pro snímky, kde je počet vajec roven **10**.

Celé řešení – příprava dat, trénování neuronové sítě, testování funkčnosti a měření přesnosti budeme implementovat v programovacím jazyce Python 3.

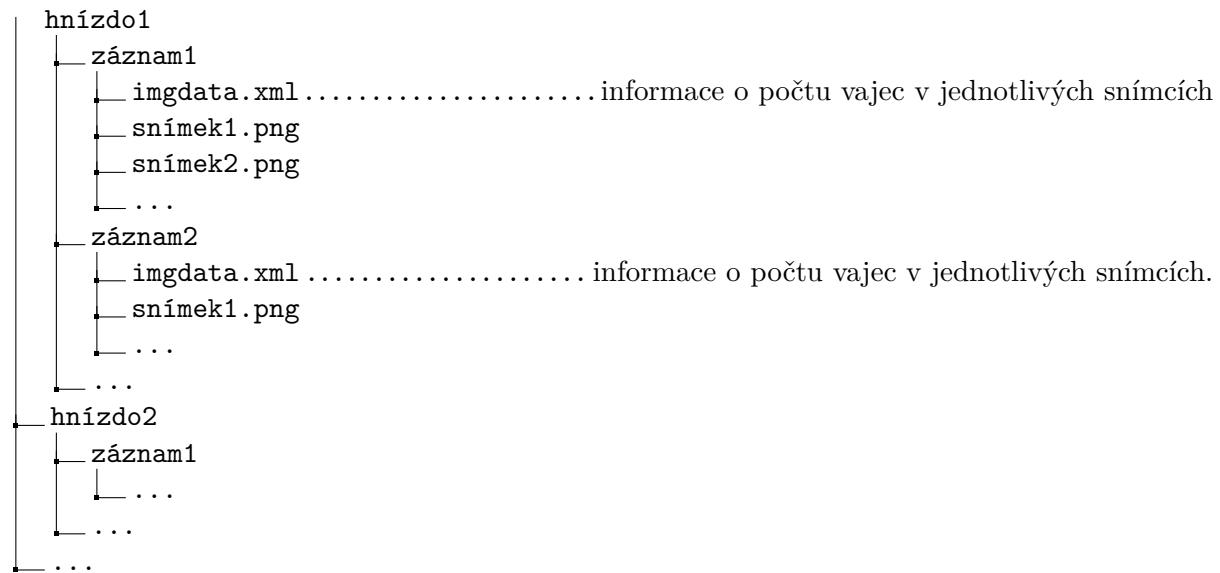
6.1 Příprava dat

Nástrojem Tagger (viz kapitola C) jsem označil **6178 snímků**. Všechny tyto snímky budou sloužit jako trénovací nebo testovací data pro tuto kapitolu.

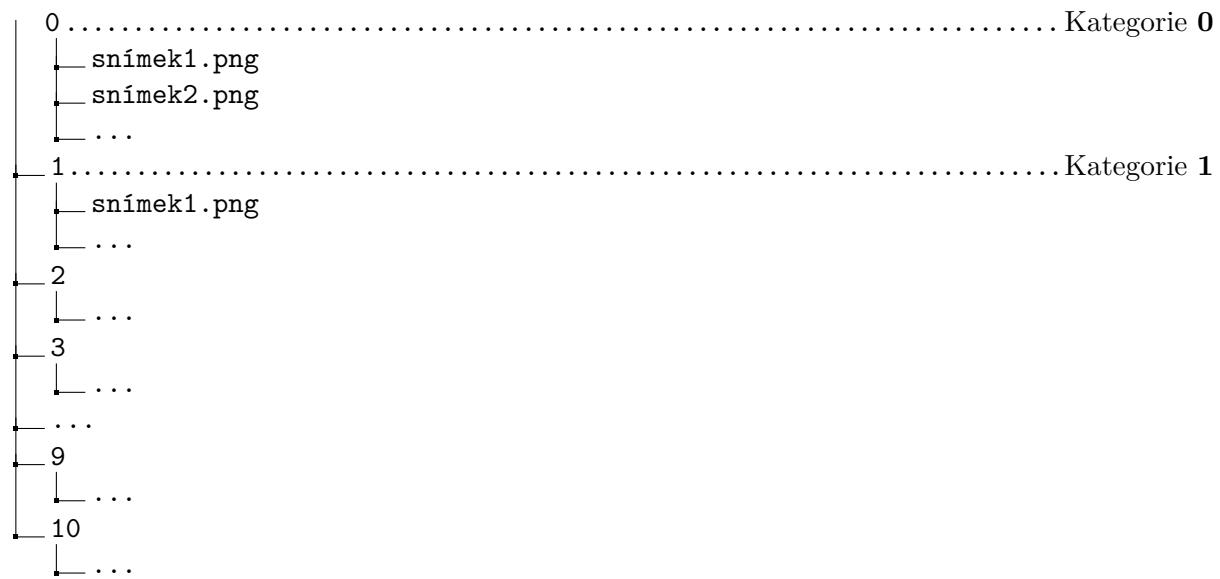
6. IMPLEMENTACE ROZPOZNÁVÁNÍ OBRAZU

Abychom nemuseli psát vlastní skripty pro trénování neuronové sítě, ale mohli použít skripty standartně dostupné [14], musíme upravit strukturu našich dat.

Současná struktura našich dat je následující:



Nová struktura dat, které potřebujeme docílit:



Každá kategorie má vlastní složku. Do každé kategorie patří snímky s počtem vajec, který odpovídá dané kategorii. Jakmile máme naše trénovací data uspořádaná do požadované struktury, je vše připraveno pro trénování neuronové sítě.

6.2 Trénování neuronové sítě

Moderní modely pro rozpoznávání obrazu mají miliony parametrů a je **extrémně** výpočetně náročné je vytrénovat. Učení „přenosem modelu“¹² je technika, která ušetří spoustu práce využitím již před-trénovaného modelu a přetrénováním pouze finálních vrstev [15]. Více informací k efektivitě tohoto řešení viz kapitola 2.

Předpokladem pro trénování neuronové sítě je nainstalovaná knihovna TensorFlow a všechny její závislosti [16]. Model, ze kterého budeme vycházet je **Inception-v4**¹³ [17], který byl vytrénován společností Google na přibližně 1,2 mil. snímků[18]. V soutěži ImageNet [19] drží model Inception-v4 nejlepšího skóre: Top-1 Accuracy¹⁴ 80.2% a Top-5 Accuracy¹⁵ 95.2% [20].

Googlem poskytovaný skript pro přetrénování finálních vrstev nepodporuje nejnovější model Inception-v4. Stačí však pár modifikací a můžeme nový model použít. Upravený skript se nachází v příloze E.

6.2.1 Struktura aplikace

Na disku vytvoříme složku `egg_recognition`, ve které se budeme pohybovat. Budeme potřebovat tuto strukturu:

```
egg_recognition ..... pracovní složka
  └── bottlenecks
  └── models
    └── training_summaries
    └── result ..... umístění výsledného modelu
    └── training_data ..... trénovací data ve formátu specifikovaném výše
      └── models
        └── inception_v4.pb ..... nutné pro správné fungování skriptu retrain.py
```

Poté spustíme připravený skript `retrain.py`:

```
1 cd egg_recognition
2 python3 retrain.py \
3   --bottleneck_dir=bottlenecks \
4   --how_many_training_steps=8000 \
5   --model_dir=models/ \
6   --summaries_dir=training_summaries/ \
7   --output_graph=result/egg_classifier_graph.pb \
8   --output_labels=result/egg_classifier_labels.txt \
9   --image_dir=training_data \
10  --print_misclassified_test_images
11  --random_crop=16
12  --random_scale=7
13  --random_brightness=4
```

Hodnoty parametrů `random_crop` a `random_scale` a `random_brightness` a `how_many_training_steps` můžeme změnit. K hodnotám uvedeným výše jsem dospěl opakováním testováním a tyto hod-

¹²Transfer learning.

¹³Dostupný ke stažení na: http://download.tensorflow.org/models/inception_v4_2016_09_09.tar.gz

¹⁴Odpověď modelu (ta s nejvyšší pravděpodobností) přesně odpovídala očekávanému výsledku.

¹⁵Kterákoli z 5 nejpravděpodobnějších odpovědí modelu odpovídala očekávanému výsledku.

6. IMPLEMENTACE ROZPOZNÁVÁNÍ OBRAZU

noty produkovaly nejlepší výsledky. Samotný skript `retrain.py` [14] poskytuje nejlepší dokumentaci dostupných parametrů.

6.3 Ověření funkčnosti a výsledky

Vytrénovaný a vyexportovaný model můžeme otestovat pomocí veřejně dostupného skriptu `label_image.py` [21]:

```
1 cd egg_recognition
2 python3 ~/tensorflow/examples/image_retraining/label_image.py \
3   --graph=result/egg_classifier_graph.pb \
4   --labels=/tmp/output_labels.txt \
5   --output_layer=final_result:0 \
6   --image=test_image.jpg
```

Výsledek je prezentován v následujícím formátu (hodnoty mohou být odlišné):

```
1 5 (score = 0.62071)
2 4 (score = 0.44595)
3 6 (score = 0.43252)
4 0 (score = 0.43049)
5 9 (score = 0.00032)
```

6.4 Závěr

Dokázali jsme, že řešení funguje. Při hromadném testu dat zjistíme, že tato implementace **není** příliš efektivní i přes to, že náš trénovací vzorek dat je poměrně velký (**6178 snímků**). Už ze samotného výsledku při vyhodnocení pouze jednoho snímku je vidět, že si síť není jistá, do jaké kategorie daný snímek zařadit. Všechny výsledky mají buď relativně nízké skóre nebo naopak všechny poměrně vysoké. Bohužel se nedostáváme k výsledku, u kterého by byla jedna kategorie dominantní¹⁶.

Největší problém spočívá v **malých rozdílech mezi jednotlivými kategoriemi**. Vajíčka tvoří pouze malou část snímku, takže rozdíl mezi jedním nebo dvěma vajíčky je malý. **Většina plochy snímku se stává šumem**, který „mate“ tuto implementaci. Kdyby kategorie reprezentovaly dvě velmi odlišné věci, jako například auto a pes, bylo by výrazně snažší snímek mezi tyto dvě dramaticky odlišné kategorie zařadit.

Řešení by se stalo mnohem efektivnější v případě, kdyby snímky, které síť zpracovává byly předem zpracované a upravené. Normalizace jasu a především ořezání snímku tak, aby na něm zbyly pouze vajíčka, by dramaticky zvýšila efektivitu této implementace.

Tato kapitola neposkytla výsledky, ve které jsem doufal. Je zde ale prostor pro vylepšení, které by toto řešení mohli udělat efektivnější než detekce objektů popsaná v kapitole 5.

¹⁶Sít by jedné kategorii přiřadila vysoké skóre (> 80%) a zbytek kategorií by mělo skóre nízké.

Implementace Java knihovny

- 7.1 Srovnání implementací neuronových sítí a volba řešení**
- 7.2 Použití exportovaného modelu v Javě**
- 7.3 Uživatelské rozhraní knihovny**
- 7.4 Implementace??**
- 7.5 Maven**
- 7.6 Výsledek**

Ověření implementace

- 8.0.1 Metodika vývoje
- 8.0.2 Testování
- 8.0.3 Možnosti vylepšení

Text

Závěr

Literatura

- [1] Česká zemědělská univerzita v Praze, Fakulta životního prostředí: *O projektu ptáci online*. [online]. [cit. 2017-12-27]. Dostupné z: <http://www.ptacionline.cz/o-projektu/o-projektu>
- [2] Česká zemědělská univerzita v Praze, Fakulta životního prostředí: *Ptáci online v médiích*. [online]. [cit. 2017-12-27]. Dostupné z: <http://www.ptacionline.cz/o-projektu>
- [3] Oracle: *jar-The Java Archive Tool*. [online]. [cit. 2017-11-14]. Dostupné z: <https://docs.oracle.com/javase/7/docs/technotes/tools/windows/jar.html>
- [4] The Apache Software Foundation: *Maven* [online]. 2017. Dostupné z: <https://maven.apache.org/>
- [5] Ministerstvo životního prostředí: *Ptáci online: Sledujte záběry z „chytré ptačí budky“ na budově MŽP* [online]. [cit. 2017-11-18]. Dostupné z: http://www.mzp.cz/cz/news_160608_Ptaci_online
- [6] Šuma, P.: *Detekce počtu mláďat v ptačích hnizdech za pomoci nástrojů rozpoznání obrazu*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.
- [7] Google Inc.: *Image Recognition* [online]. [cit. 2017-11-17]. Dostupné z: https://www.tensorflow.org/tutorials/image_recognition
- [8] Google Inc.: *Tensorflow API Documentation* [online]. [cit. 2017-11-17]. Dostupné z: https://www.tensorflow.org/api_docs/
- [9] Google Inc.: *Getting Started with TensorFlow* [online]. [cit. 2017-11-17]. Dostupné z: https://www.tensorflow.org/get_started/
- [10] Google Inc.: *Tensorflow Object Detection API* [online]. [cit. 2017-11-17]. Dostupné z: https://github.com/tensorflow/models/tree/master/research/object_detection
- [11] Google Inc.: *TensorFlow: Importing Data* [online]. [cit. 2017-11-17]. Dostupné z: https://www.tensorflow.org/programmers_guide/datasets
- [12] Pivotal Software, Inc.: *Spring Boot* [online]. 2018. Dostupné z: <https://projects.spring.io/spring-boot/>

LITERATURA

- [13] Tzutalin: *LabelImg [online]*. Git code (2015). Dostupné z: <https://github.com/tzutalin/labelImg>
- [14] Google Inc.: *Simple transfer learning with Inception v3 or Mobilenet models. [online]*. [cit. 2017-12-03]. Dostupné z: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/image_retraining/retrain.py
- [15] Donahue, J.; Jia, Y.; Vinyals, O.; aj.: DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. Technická zpráva, UC Berkeley & ICSI, 2013, [cit. 2017-12-08]. Dostupné z: <https://arxiv.org/pdf/1310.1531v1.pdf>
- [16] Google Inc.: *Installing TensorFlow [online]*. [cit. 2017-12-06]. Dostupné z: <https://www.tensorflow.org/install/>
- [17] Szegedy, C.; Ioffe, S.; Vanhoucke, V.; aj.: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. Technická zpráva, Google Inc., 2016, [cit. 2017-12-09]. Dostupné z: <https://arxiv.org/pdf/1512.00567.pdf>
- [18] Google Inc.: *Preparing the datasets [online]*. [cit. 2017-12-12]. Dostupné z: <https://github.com/tensorflow/models/tree/master/research/slim#preparing-the-datasets>
- [19] Stanford Vision Lab, Stanford University, Princeton University: *ImageNet [online]*. [cit. 2017-12-06]. Dostupné z: <http://www.image-net.org/>
- [20] Russakovsky, O.; Deng, J.; Su, H.; aj.: ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, ročník 115, č. 3, 2015: s. 211–252, doi: 10.1007/s11263-015-0816-y.
- [21] Google Inc.: *Using the Retrained Model. [online]*. [cit. 2017-12-03]. Dostupné z: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/label_image/label_image.py
- [22] Oracle: *Javadoc tool. [online]*. [cit. 2017-11-12]. Dostupné z: <http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html>
- [23] *TexDoclet [online]*. 2017. Dostupné z: <http://doclet.github.io/>

Seznam použitých zkratek

GUI Graphical user interface

XML Extensible markup language

HTML HyperText Markup Language

RAM Random-access memory

PDF Portable Document Format

NN Neural Network

CNN Convolutional Neural Network

RCNN Recursive Convolutional Neural Network

API Application Programming Interface

PNG Portable Network Graphics

JPEG Joint Photographic Experts Group

Dokumentace API knihovny

Příloha obsahuje dokumentaci API Java knihovny, která je výsledkem této práce. Dokumentace byla vygenerována ze zdrojových kódů nástroji Javadoc[22] a TexDoclet[23]. Kód je psán v anglickém jazyce, stejně jako jeho dokumentace. Proto je text i zde v angličtině.

B.1 Package org.cvut.havluja1.eggdetector

<i>Package Contents</i>	<i>Page</i>
Classes	
EggDetector	53
SequenceClassifier	57

B.1.1 Class EggDetector

B.1.1.1 Count the number of eggs in given images

The egg detector is a library that helps you count the number of eggs in a given folder. Egg detector works by using TeonsorFlow Object Detection API in the background. To learn more, see <https://www.tensorflow.com>.

Example usage:

```
1 EggDetector eggDetector = new EggDetector();
2 SequenceClassifier sequenceClassifier = eggDetector.evaluate(new File("image_dir"));
3 System.out.println("final count: " + sequenceClassifier.getFinalCount());
4 System.out.println("individual scores: " + sequenceClassifier.getIndividualCounts());
5 eggDetector.closeSession();
```

B. DOKUMENTACE API KNIHOVNY

B.1.1.2 Declaration

```
1 public class EggDetector  
2 extends java.lang.Object
```

B.1.1.3 Constructor summary

EggDetector()

Constructor loads the pre-trained frozen graph into memory.

B.1.1.4 Method summary

closeSession()

Closes the EggDetector session.

evaluate(File)

Runs egg detection on a given *dir*.

getMinimalConfidence()

Get the minimalConfidence setting for this instance.

isDebugMode()

Get this instance's debug mode setting.

setDebugMode(boolean)

Set this instance's debug mode setting.

setMinimalConfidence(float)

Set the minimalConfidence setting for this instance.

toString()

B.1.1.5 Constructors

- *EggDetector*

```
1 public EggDetector()
```

– Description

Constructor loads the pre-trained frozen graph into memory.

It also checks whether TensorFlow is supported on your platform.

B.1.1.6 Methods

- *closeSession*

```
1 public void closeSession() throws java.lang.IllegalStateException
```

- **Description**
Closes the EggDetector session. This instance of EggDetector will not be usable again.
- **Throws**
 - * `java.lang.IllegalStateException` – if the session has been closed already by calling `closeSession()`
- *evaluate*

```
1 public SequenceClassifier evaluate(java.io.File dir) throws  
    java.lang.IllegalArgumentException, java.lang.IllegalStateException
```

- **Description**
Runs egg detection on a given `dir`.
- **Parameters**
 - * `dir` – a directory containing .jpg or .png files for object detection
- **Returns** –
- **Throws**
 - * `java.lang.IllegalArgumentException` – if `dir` is not a directory or contains no images
 - * `java.lang.IllegalStateException` – if the session has been closed already by calling `closeSession()`
- *getMinimalConfidence*

```
1 public float getMinimalConfidence()
```

- **Description**
Get the `minimalConfidence` setting for this instance.
Minimal Confidence score is used as a confidence boundary during the process of object detection. An object that has been detected with a confidence score lower than `minimalConfidence` is ignored. An object that has been detected with a confidence score higher or equal than `minimalConfidence` is added to the final result list.
- **Returns** – This instance's `minimalConfidence` setting.

- *isDebugEnabled*

```
1 public boolean isDebugEnabled()
```

- **Description**
Get this instance's debug mode setting.
If debug mode is enabled (set to true), the library will open a `JFrame` for each processed image with detections graphically highlighted.

B. DOKUMENTACE API KNIHOVNY

- Returns – debug mode setting for this instance
- *setDebugMode*

```
1 public void setDebugMode(boolean debugMode) throws  
    java.lang.IllegalStateException
```

- Description

Set this instance's debug mode setting.
If debug mode is enabled (set to true), the library will open a `JFrame` for each processed image with detections graphically highlighted.
- Parameters
 - * `debugMode` – turn the debug mode on or off
- Throws
 - * `java.lang.IllegalStateException` – if the session has been closed already by calling `closeSession()`
- *setMinimalConfidence*

```
1 public void setMinimalConfidence(float minimalConfidence) throws  
    java.lang.IllegalStateException
```

- Description

Set the *minimalConfidence* setting for this instance.
Minimal Confidence score is used as a confidence boundary during the process of object detection. An object that has been detected with a confidence score lower than *minimalConfidence* is ignored. An object that has been detected with a confidence score higher or equal than *minimalConfidence* is added to the final result list.
- Parameters
 - * `minimalConfidence` – *minimalConfidence* for this instance
- Throws
 - * `java.lang.IllegalStateException` – if the session has been closed already by calling `closeSession()`
- *toString*

```
1 public java.lang.String toString()
```

B.1.2 Class SequenceClassifier

B.1.2.1 A class containing object detection results for a given directory

SequenceClassifier is a data class containing the results of object detection for a given directory. When constructed, object detection is performed on all images and results are stored in memory.

Example usage:

```
1 EggDetector eggDetector = new EggDetector();
2 SequenceClassifier sequenceClassifier = eggDetector.evaluate(new File("image_dir"));
3 System.out.println("final count: " + sequenceClassifier.getFinalCount());
4 System.out.println("individual scores: " + sequenceClassifier.getIndividualCounts());
5 eggDetector.closeSession();
```

B.1.2.2 Declaration

```
1 public class SequenceClassifier
2     extends java.lang.Object
```

B.1.2.3 Method summary

getFinalCount()

Get the final score for the entire directory.

getIndividualCounts()

Gets the individual egg count for every image provided.

B.1.2.4 Methods

- *getFinalCount*

```
1 public java.lang.Integer getFinalCount()
```

– Description

Get the final score for the entire directory.

The final score is calculated as follows:

- * individual scores of images are sorted and counted
- * the highest egg count is returned as a result if we detected this egg count in at least two different images
- * if no two images contain the same egg count, the highest detected egg count is returned
- * if no eggs are detected in any of the images, 0 is returned

– Returns – final egg count for this instance

- *getIndividualCounts*

```
1 public java.util.Map getIndividualCounts()
```

B. DOKUMENTACE API KNIHOVNY

- **Description**
Gets the individual egg count for every image provided.
- **Returns** – A map of individual scores. The key is the filename. The value is the egg count.

Tagger

Tato příloha obsahuje zdrojový kód a způsob použití nástroje Tagger. Kompletní dokumentace a zdrojový kód je dostupný na přiloženém CD (viz příloha G).

C.1 Dokumentace (v AJ)

Data tagger is a simple web application used for data tagging used to train a neural network. The data strictly needs to follow the structure of <http://athena.pef.czu.cz/ptacionline/134572snaps/>

C.1.1 Usage manual

- Compile (if needed) the app with `mvn clean package`
- Run with `target/run.sh` on Linux or `target\run.bat` on Windows. Both scripts accept data location as the first and only parameter.
- For example: `target/run.sh /home/test/egg/data/nest1`

C.2 Zdrojový kód

Zdrojový kód je dostupný na přiloženém CD (viz příloha G).

C.3 Ukázkový výstup

Po určení počtu vajec v každém snímku je do příslušné složky vygenerován soubor `imgdata.xml`, který je použit pro trénování a později testování neuronové sítě. Níže je přiložen ukázkový výpis souboru.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <java version="1.8.0_121" class="java.beans.XMLDecoder">
3 <object class="org.cvut.havlaja1.tagger.model.FolderData">
4   <void property="imgData">
5     <object class="java.util.ArrayList">
6       <void method="add">
```

C. TAGGER

```
7   <object class="org.cvut.havlujai.tagger.model.ImgData">
8     <void property="eggCount">
9       <int>0</int>
10    </void>
11    <void property="name">
12      <string>snap0001-0000000668.png</string>
13    </void>
14  </object>
15 </void>
16 <void method="add">
17   <object class="org.cvut.havlujai.tagger.model.ImgData">
18     <void property="eggCount">
19       <int>2</int>
20     </void>
21     <void property="name">
22       <string>snap0002-0000001419.png</string>
23     </void>
24   </object>
25 </void>
26 <void method="add">
27   <object class="org.cvut.havlujai.tagger.model.ImgData">
28     <void property="eggCount">
29       <int>2</int>
30     </void>
31     <void property="name">
32       <string>snap0003-0000002341.png</string>
33     </void>
34   </object>
35 </void>
36 <void method="add">
37   <object class="org.cvut.havlujai.tagger.model.ImgData">
38     <void property="eggCount">
39       <int>8</int>
40     </void>
41     <void property="name">
42       <string>snap0004-0000003160.png</string>
43     </void>
44   </object>
45 </void>
46 <void method="add">
47   <object class="org.cvut.havlujai.tagger.model.ImgData">
48     <void property="eggCount">
49       <int>8</int>
50     </void>
51     <void property="name">
52       <string>snap0005-0000004082.png</string>
53     </void>
54   </object>
55 </void>
56 </object>
57 </void>
58 </object>
59 </java>
```

Folder Trimmer

Tato příloha obsahuje zdrojový kód a způsob použití nástroje **FolderTrimmer**. Kompletní dokumentace a zdrojový kód je dostupný na přiloženém CD (viz příloha G).

D.1 Dokumentace (v AJ)

Folder trimmer is a command line tool for deleting useless data the download script downloads.

D.1.1 What does it do

- Deletes files that do not end on either `.png`, `.xml`, or `.txt`.
- If a folder does not contain another folder or at least one of the files listed above, the folder is deleted.

D.1.2 Usage manual

- Compile (if needed) the app with `mvn clean package`.
- Run with `target/run.sh` on Linux or `target\run.bat` on Windows. Both scripts accept two parameters:
 - First parameter can be `true` or `false` and it determines whether you want to delete everything except manually tagged data (containing `imgdata.xml`) in case of `true` or if you just want to delete the useless data in case of `false`.
 - Second parameter specifies the root folder of the data that should be trimmed.
 - For example to delete only useless data but keep all the folders with some image data, use: `target\run.bat false C:\school\bakalarka\data`.
 - To delete everything except tagged data, run: `target\run.bat true C:\school\bakalarka\data`.

D.2 Zdrojový kód

D. FOLDER TRIMMER

Algoritmus D.1: Zdrojový kód nástroje FolderTrimmer

```
1 package org.cvut.havlujal.foldertrimmer;
2
3 import java.io.File;
4 import java.io.IOException;
5
6 import org.apache.commons.io.FileUtils;
7 import org.apache.commons.io.FilenameUtils;
8
9 public class FolderTrimmer {
10     public static void main(String[] args) throws IOException {
11         File rootDir = new File(args[0]);
12
13         if (!rootDir.exists() || !rootDir.isDirectory()) {
14             throw new IllegalArgumentException("root dir does not exist");
15         }
16
17         System.out.println("finding useless data...");
18         findAndDeleteEmptyDirs(rootDir);
19         System.out.println("done");
20     }
21
22     private static void findAndDeleteEmptyDirs(File dir) {
23         final boolean[] shouldBeDeleted = {true};
24         final boolean leaveOnlyTaggedData = System.getProperty("leaveonlytagged").equalsIgnoreCase("true");
25
26         File[] toBeProcessed = dir.listFiles((file, s) -> {
27             File workingFile = new File(file, s);
28
29             // if dir -> return true and tag this folder not to be deleted
30             if (workingFile.isDirectory()) {
31                 shouldBeDeleted[0] = false;
32                 return true;
33             }
34
35             if (workingFile.isFile()) {
36                 if (leaveOnlyTaggedData) { // if we want to keep only tagged data
37                     if (s.equals("imgdata.xml")) {
38                         shouldBeDeleted[0] = false;
39                         return false;
40                     } else {
41                         if (FilenameUtils.getExtension(s).equals("png")) {
42                             return false;
43                         }
44                         return true;
45                     }
46                 } else { // If file and is not xml, txt or png return true. If it is, tag this folder not to be deleted.
47                     if (FilenameUtils.getExtension(s).equals("xml")
48                         || FilenameUtils.getExtension(s).equals("png")
49                         || FilenameUtils.getExtension(s).equals("txt")) {
50                         shouldBeDeleted[0] = false;
51                         return false;
52                     } else {
53                         return true;
54                     }
55                 }
56             }
57         });
58
59         return true;
60     });
61
62     if (shouldBeDeleted[0]) {
63         try {
64             FileUtils.deleteDirectory(dir);
65             System.out.println("[D] deleting dir: " + dir.getAbsolutePath());
66         } catch (IOException e) {
67             e.printStackTrace();
68         }
69     } else {
70         if (toBeProcessed.length > 0) {
71             for (File currFile : toBeProcessed) {
72                 // if file -> delete
73                 if (currFile.isFile()) {
74                     if (currFile.delete()) {
75                         System.out.println("[F] deleting file: " + currFile.getAbsolutePath());
76                     }
77                     continue;
78                 }
79             }
80         }
81     }
82 }
```

```
78         }
79         // if dir -> recursive call
80         if (currFile.isDirectory()) {
81             findAndDeleteEmptyDirs(currFile);
82         }
83     }
84 }
85 }
86 }
87 }
88 }
```

Přetrénování finální vrstvy modelu Inception-v4

Tato příloha obsahuje zdrojový kód skriptu `retrain.py`, který je použit pro trénování neuronové sítě v kapitole 6.

Aby skript správně fungoval, je nutné, aby ve složce s modelem byl soubor https://deepdetect.com/models/tf/inception_v4.pb.

Algoritmus E.1: Přetrénování finální vrstvy modelu Inception-v4.

```
1 from __future__ import absolute_import
2 from __future__ import division
3 from __future__ import print_function
4
5 import argparse
6 from datetime import datetime
7 import hashlib
8 import os.path
9 import random
10 import re
11 import sys
12 import tarfile
13
14 import numpy as np
15 from six.moves import urllib
16 import tensorflow as tf
17
18 from tensorflow.contrib.quantize.python import quant_ops
19 from tensorflow.python.framework import graph_util
20 from tensorflow.python.framework import tensor_shape
21 from tensorflow.python.platform import gfile
22 from tensorflow.python.util import compat
23
24 FLAGS = None
25
26 MAX_NUM_IMAGES_PER_CLASS = 2 ** 27 - 1 # ~134M
27
28
29 def create_image_lists(image_dir, testing_percentage, validation_percentage):
30     if not gfile.Exists(image_dir):
31         tf.logging.error("Image directory '" + image_dir + "' not found.")
32         return None
33     result = {}
34     sub_dirs = [x[0] for x in gfile.Walk(image_dir)]
35     # The root directory comes first, so skip it.
36     is_root_dir = True
37     for sub_dir in sub_dirs:
38         if is_root_dir:
39             is_root_dir = False
40             continue
41     extensions = ['jpg', 'jpeg', 'JPG', 'JPEG']
42     file_list = []
```

E. PŘETRÉNOVÁNÍ FINÁLNÍ VRSTVY MODELU INCEPTION-V4

```

43     dir_name = os.path.basename(sub_dir)
44     if dir_name == image_dir:
45         continue
46     tf.logging.info("Looking for images in '" + dir_name + "'")
47     for extension in extensions:
48         file_glob = os.path.join(image_dir, dir_name, '*' + extension)
49         file_list .extend(gfile .Glob(file_glob))
50     if not file_list :
51         tf.logging.warning('No files found')
52     continue
53     if len( file_list ) < 20:
54         tf.logging.warning(
55             'WARNING: Folder has less than 20 images, which may cause issues.')
56     elif len( file_list ) > MAX_NUM_IMAGES_PER_CLASS:
57         tf.logging.warning(
58             'WARNING: Folder {} has more than {} images. Some images will ,
59             'never be selected.'.format(dir_name, MAX_NUM_IMAGES_PER_CLASS))
60     label_name = re.sub(r'[^a-z0-9]+', ' ', dir_name.lower())
61     training_images = []
62     testing_images = []
63     validation_images = []
64     for file_name in file_list :
65         base_name = os.path.basename(file_name)
66         hash_name = re.sub(r'_nohash_\w+$', '', file_name)
67         hash_name_hashed = hashlib.sha1(compat.as_bytes(hash_name)).hexdigest()
68         percentage_hash = ((int(hash_name_hashed, 16) *
69             (MAX_NUM_IMAGES_PER_CLASS + 1)) *
70             (100.0 / MAX_NUM_IMAGES_PER_CLASS))
71         if percentage_hash < validation_percentage:
72             validation_images.append(base_name)
73         elif percentage_hash < (testing_percentage + validation_percentage):
74             testing_images.append(base_name)
75         else :
76             training_images.append(base_name)
77     result [label_name] = {
78         'dir': dir_name,
79         'training': training_images,
80         'testing': testing_images,
81         'validation': validation_images,
82     }
83     return result
84
85
86 def get_image_path(image_lists, label_name, index, image_dir, category):
87     if label_name not in image_lists:
88         tf.logging.fatal('Label does not exist %s.', label_name)
89     label_lists = image_lists[label_name]
90     if category not in label_lists:
91         tf.logging.fatal('Category does not exist %s.', category)
92     category_list = label_lists[category]
93     if not category_list:
94         tf.logging.fatal('Label %s has no images in the category %s.',
95                         label_name, category)
96     mod_index = index % len(category_list)
97     base_name = category_list[mod_index]
98     sub_dir = label_lists['dir']
99     full_path = os.path.join(image_dir, sub_dir, base_name)
100    return full_path
101
102
103 def get_bottleneck_path(image_lists, label_name, index, bottleneck_dir,
104                         category, architecture):
105     return get_image_path(image_lists, label_name, index, bottleneck_dir,
106                          category) + '_' + architecture + '.txt'
107
108
109 def create_model_graph(model_info):
110     with tf.Graph().as_default() as graph:
111         model_path = os.path.join(FLAGS.model_dir, model_info['model_file_name'])
112         print('Model path: ', model_path)
113         with gfile.FastGFile(model_path, 'rb') as f:
114             graph_def = tf.GraphDef()
115             graph_def.ParseFromString(f.read())
116             bottleneck_tensor, resized_input_tensor = (tf.import_graph_def(
117                 graph_def,
118                 name='',
119                 return_elements=[

120                     model_info['bottleneck_tensor_name'],
121                     model_info['resized_input_tensor_name'],

```

```

122     ]))
123     return graph, bottleneck_tensor, resized_input_tensor
124
125
126 def run_bottleneck_on_image(sess, image_data, image_data_tensor,
127                             decoded_image_tensor, resized_input_tensor,
128                             bottleneck_tensor):
129     # First decode the JPEG image, resize it, and rescale the pixel values.
130     resized_input_values = sess.run(decoded_image_tensor,
131                                    {image_data_tensor: image_data})
132     # Then run it through the recognition network.
133     bottleneck_values = sess.run(bottleneck_tensor,
134                                 {resized_input_tensor: resized_input_values})
135     bottleneck_values = np.squeeze(bottleneck_values)
136     return bottleneck_values
137
138
139 def maybe_download_and_extract(data_url):
140     dest_directory = FLAGS.model_dir
141     if not os.path.exists(dest_directory):
142         os.makedirs(dest_directory)
143     filename = data_url.split('/')[-1]
144     filepath = os.path.join(dest_directory, filename)
145     if not os.path.exists(filepath):
146
147         def _progress(count, block_size, total_size):
148             sys.stdout.write('\r>> Downloading %s %.1f%%' %
149                           (filename,
150                            float(count * block_size) / float(total_size) * 100.0))
151         sys.stdout.flush()
152
153     filepath, _ = urllib.request.urlretrieve(data_url, filepath, _progress)
154     print()
155     statinfo = os.stat(filepath)
156     tf.logging.info('Successfully downloaded', filename, statinfo.st_size,
157                     'bytes.')
158     print('Extracting file from ', filepath)
159     tarfile.open(filepath, 'r:gz').extractall(dest_directory)
160 else:
161     print('Not extracting or downloading files, model already present in disk')
162
163
164 def ensure_dir_exists(dir_name):
165     if not os.path.exists(dir_name):
166         os.makedirs(dir_name)
167
168
169 bottleneck_path_2_bottleneck_values = {}
170
171
172 def create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
173                            image_dir, category, sess, jpeg_data_tensor,
174                            decoded_image_tensor, resized_input_tensor,
175                            bottleneck_tensor):
176     tf.logging.info('Creating bottleneck at ' + bottleneck_path)
177     image_path = get_image_path(image_lists, label_name, index,
178                                image_dir, category)
179     if not gfile.Exists(image_path):
180         tf.logging.fatal('File does not exist %s', image_path)
181     image_data = gfile.FastGFile(image_path, 'rb').read()
182     try:
183         bottleneck_values = run_bottleneck_on_image(
184             sess, image_data, jpeg_data_tensor, decoded_image_tensor,
185             resized_input_tensor, bottleneck_tensor)
186     except Exception as e:
187         raise RuntimeError('Error during processing file %s (%s)' % (image_path,
188                                                                       str(e)))
189     bottleneck_string = ','.join(str(x) for x in bottleneck_values)
190     with open(bottleneck_path, 'w') as bottleneck_file:
191         bottleneck_file.write(bottleneck_string)
192
193
194 def get_or_create_bottleneck(sess, image_lists, label_name, index, image_dir,
195                             category, bottleneck_dir, jpeg_data_tensor,
196                             decoded_image_tensor, resized_input_tensor,
197                             bottleneck_tensor, architecture):
198     label_lists = image_lists[label_name]
199     sub_dir = label_lists['dir']
200     sub_dir_path = os.path.join(bottleneck_dir, sub_dir)

```

E. PŘETRÉNOVÁNÍ FINÁLNÍ VRSTVY MODELU INCEPTION-V4

```
201 ensure_dir_exists(sub_dir_path)
202 bottleneck_path = get_bottleneck_path(image_lists, label_name, index,
203                                         bottleneck_dir, category, architecture)
204 if not os.path.exists(bottleneck_path):
205     create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
206                             image_dir, category, sess, jpeg_data_tensor,
207                             decoded_image_tensor, resized_input_tensor,
208                             bottleneck_tensor)
209 with open(bottleneck_path, 'r') as bottleneck_file:
210     bottleneck_string = bottleneck_file.read()
211 did_hit_error = False
212 try:
213     bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
214 except ValueError:
215     tf.logging.warning('Invalid float found, recreating bottleneck')
216     did_hit_error = True
217 if did_hit_error:
218     create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
219                             image_dir, category, sess, jpeg_data_tensor,
220                             decoded_image_tensor, resized_input_tensor,
221                             bottleneck_tensor)
222 with open(bottleneck_path, 'r') as bottleneck_file:
223     bottleneck_string = bottleneck_file.read()
224 # Allow exceptions to propagate here, since they shouldn't happen after a
225 # fresh creation
226     bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
227 return bottleneck_values
228
229
230 def cache_bottlenecks(sess, image_lists, image_dir, bottleneck_dir,
231                       jpeg_data_tensor, decoded_image_tensor,
232                       resized_input_tensor, bottleneck_tensor, architecture):
233     how_many_bottlenecks = 0
234     ensure_dir_exists(bottleneck_dir)
235     for label_name, label_lists in image_lists.items():
236         for category in ['training', 'testing', 'validation']:
237             category_list = label_lists[category]
238             for index, unused_base_name in enumerate(category_list):
239                 get_or_create_bottleneck(
240                     sess, image_lists, label_name, index, image_dir, category,
241                     bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
242                     resized_input_tensor, bottleneck_tensor, architecture)
243
244     how_many_bottlenecks += 1
245     if how_many_bottlenecks % 100 == 0:
246         tf.logging.info(
247             str(how_many_bottlenecks) + ' bottleneck files created.')
248
249
250 def get_random_cached_bottlenecks(sess, image_lists, how_many, category,
251                                  bottleneck_dir, image_dir, jpeg_data_tensor,
252                                  decoded_image_tensor, resized_input_tensor,
253                                  bottleneck_tensor, architecture):
254     class_count = len(image_lists.keys())
255     bottlenecks = []
256     ground_truths = []
257     filenames = []
258     if how_many >= 0:
259         # Retrieve a random sample of bottlenecks.
260         for unused_i in range(how_many):
261             label_index = random.randrange(class_count)
262             label_name = list(image_lists.keys())[label_index]
263             image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
264             image_name = get_image_path(image_lists, label_name, image_index,
265                                         image_dir, category)
266             bottleneck = get_or_create_bottleneck(
267                 sess, image_lists, label_name, image_index, image_dir, category,
268                 bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
269                 resized_input_tensor, bottleneck_tensor, architecture)
270             bottlenecks.append(bottleneck)
271             ground_truths.append(label_index)
272             filenames.append(image_name)
273     else:
274         # Retrieve all bottlenecks.
275         for label_index, label_name in enumerate(image_lists.keys()):
276             for image_index, image_name in enumerate(
277                 image_lists[label_name][category]):
278                 image_name = get_image_path(image_lists, label_name, image_index,
279                                         image_dir, category)
```

```

280     bottleneck = get_or_create_bottleneck(
281         sess, image_lists, label_name, image_index, image_dir, category,
282         bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
283         resized_input_tensor, bottleneck_tensor, architecture)
284     bottlenecks.append(bottleneck)
285     ground_truths.append(label_index)
286     filenames.append(image_name)
287     return bottlenecks, ground_truths, filenames
288
289
290 def get_random_distorted_bottlenecks(
291     sess, image_lists, how_many, category, image_dir, input_jpeg_tensor,
292     distorted_image, resized_input_tensor, bottleneck_tensor):
293     class_count = len(image_lists.keys())
294     bottlenecks = []
295     ground_truths = []
296     for unused_i in range(how_many):
297         label_index = random.randrange(class_count)
298         label_name = list(image_lists.keys())[label_index]
299         image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
300         image_path = get_image_path(image_lists, label_name, image_index, image_dir,
301                                     category)
302         if not gfile.Exists(image_path):
303             tf.logging.fatal('File does not exist %s', image_path)
304         jpeg_data = gfile.FastGFile(image_path, 'rb').read()
305         # Note that we materialize the distorted_image_data as a numpy array before
306         # sending running inference on the image. This involves 2 memory copies and
307         # might be optimized in other implementations.
308         distorted_image_data = sess.run(distorted_image,
309                                         {input_jpeg_tensor: jpeg_data})
310         bottleneck_values = sess.run(bottleneck_tensor,
311                                      {resized_input_tensor: distorted_image_data})
312         bottleneck_values = np.squeeze(bottleneck_values)
313         bottlenecks.append(bottleneck_values)
314         ground_truths.append(label_index)
315     return bottlenecks, ground_truths
316
317
318 def should_distort_images(flip_left_right, random_crop, random_scale,
319                         random_brightness):
320     return (flip_left_right or (random_crop != 0) or (random_scale != 0) or
321             (random_brightness != 0))
322
323
324 def add_input_distortions(flip_left_right, random_crop, random_scale,
325                           random_brightness, input_width, input_height,
326                           input_depth, input_mean, input_std):
327     jpeg_data = tf.placeholder(tf.string, name='DistortJPGInput')
328     decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_depth)
329     decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32)
330     decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
331     margin_scale = 1.0 + (random_crop / 100.0)
332     resize_scale = 1.0 + (random_scale / 100.0)
333     margin_scale_value = tf.constant(margin_scale)
334     resize_scale_value = tf.random_uniform(tensor_shape.scalar(),
335                                            minval=1.0,
336                                            maxval=resize_scale)
337     scale_value = tf.multiply(margin_scale_value, resize_scale_value)
338     precrop_width = tf.multiply(scale_value, input_width)
339     precrop_height = tf.multiply(scale_value, input_height)
340     precrop_shape = tf.stack([precrop_height, precrop_width])
341     precrop_shape_as_int = tf.cast(precrop_shape, dtype=tf.int32)
342     precropped_image = tf.image.resize_bilinear(decoded_image_4d,
343                                                precrop_shape_as_int)
344     precropped_image_3d = tf.squeeze(precropped_image, squeeze_dims=[0])
345     cropped_image = tf.random_crop(precropped_image_3d,
346                                   [input_height, input_width, input_depth])
347     if flip_left_right:
348         flipped_image = tf.image.random_flip_left_right(cropped_image)
349     else:
350         flipped_image = cropped_image
351     brightness_min = 1.0 - (random_brightness / 100.0)
352     brightness_max = 1.0 + (random_brightness / 100.0)
353     brightness_value = tf.random_uniform(tensor_shape.scalar(),
354                                         minval=brightness_min,
355                                         maxval=brightness_max)
356     brightened_image = tf.multiply(flipped_image, brightness_value)
357     offset_image = tf.subtract(brightened_image, input_mean)
358     mul_image = tf.multiply(offset_image, 1.0 / input_std)

```

E. PŘETRÉNOVÁNÍ FINÁLNÍ VRSTVY MODELU INCEPTION-v4

```

359 distort_result = tf.expand_dims(mul_image, 0, name='DistortResult')
360 return jpeg_data, distort_result
361
362
363 def variable_summaries(var):
364     with tf.name_scope('summaries'):
365         mean = tf.reduce_mean(var)
366         tf.summary.scalar('mean', mean)
367         with tf.name_scope('stddev'):
368             stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
369             tf.summary.scalar('stddev', stddev)
370             tf.summary.scalar('max', tf.reduce_max(var))
371             tf.summary.scalar('min', tf.reduce_min(var))
372             tf.summary.histogram('histogram', var)
373
374
375 def add_final_training_ops(class_count, final_tensor_name, bottleneck_tensor,
376                           bottleneck_tensor_size, quantize_layer):
377     with tf.name_scope('input'):
378         bottleneck_input = tf.placeholder_with_default(
379             bottleneck_tensor,
380             shape=[None, bottleneck_tensor_size],
381             name='BottleneckInputPlaceholder')
382
383         ground_truth_input = tf.placeholder(
384             tf.int64, [None], name='GroundTruthInput')
385
386 # Organizing the following ops as 'final_training_ops' so they're easier
387 # to see in TensorBoard
388 layer_name = 'final_training_ops'
389 with tf.name_scope(layer_name):
390     with tf.name_scope('weights'):
391         initial_value = tf.truncated_normal(
392             [bottleneck_tensor_size, class_count], stddev=0.001)
393         layer_weights = tf.Variable(initial_value, name='final_weights')
394         if quantize_layer:
395             quantized_layer_weights = quant_ops.MovingAvgQuantize(
396                 layer_weights, is_training=True)
397             variable_summaries(quantized_layer_weights)
398
399         variable_summaries(layer_weights)
400     with tf.name_scope('biases'):
401         layer_biases = tf.Variable(tf.zeros([class_count]), name='final_biases')
402         if quantize_layer:
403             quantized_layer_biases = quant_ops.MovingAvgQuantize(
404                 layer_biases, is_training=True)
405             variable_summaries(quantized_layer_biases)
406
407         variable_summaries(layer_biases)
408
409     with tf.name_scope('Wx_plus_b'):
410         if quantize_layer:
411             logits = tf.matmul(bottleneck_input,
412                               quantized_layer_weights) + quantized_layer_biases
413             logits = quant_ops.MovingAvgQuantize(
414                 logits,
415                 init_min=-32.0,
416                 init_max=32.0,
417                 is_training=True,
418                 num_bits=8,
419                 narrow_range=False,
420                 ema_decay=0.5)
421             tf.summary.histogram('pre_activations', logits)
422         else:
423             logits = tf.matmul(bottleneck_input, layer_weights) + layer_biases
424             tf.summary.histogram('pre_activations', logits)
425
426         final_tensor = tf.nn.softmax(logits, name=final_tensor_name)
427
428         tf.summary.histogram('activations', final_tensor)
429
430     with tf.name_scope('cross_entropy'):
431         cross_entropy_mean = tf.losses.sparse_softmax_cross_entropy(
432             labels=ground_truth_input, logits=logits)
433
434         tf.summary.scalar('cross_entropy', cross_entropy_mean)
435
436     with tf.name_scope('train'):
437         optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning_rate)

```

```

438     train_step = optimizer.minimize(cross_entropy_mean)
439
440     return (train_step, cross_entropy_mean, bottleneck_input, ground_truth_input,
441             final_tensor)
442
443
444 def add_evaluation_step(result_tensor, ground_truth_tensor):
445     with tf.name_scope('accuracy'):
446         with tf.name_scope('correct_prediction'):
447             prediction = tf.argmax(result_tensor, 1)
448             correct_prediction = tf.equal(prediction, ground_truth_tensor)
449             with tf.name_scope('accuracy'):
450                 evaluation_step = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
451             tf.summary.scalar('accuracy', evaluation_step)
452     return evaluation_step, prediction
453
454
455 def save_graph_to_file(sess, graph, graph_file_name):
456     output_graph_def = graph_util.convert_variables_to_constants(
457         sess, graph.as_graph_def(), [FLAGS.final_tensor_name])
458
459     with gfile.FastGFile(graph_file_name, 'wb') as f:
460         f.write(output_graph_def.SerializeToString())
461     return
462
463
464 def prepare_file_system():
465     # Setup the directory we'll write summaries to for TensorBoard
466     if tf.gfile.Exists(FLAGS.summaries_dir):
467         tf.gfile.DeleteRecursively(FLAGS.summaries_dir)
468     tf.gfile.MakeDirs(FLAGS.summaries_dir)
469     if FLAGS.intermediate_store_frequency > 0:
470         ensure_dir_exists(FLAGS.intermediate_output_graphs_dir)
471     return
472
473
474 def create_model_info(architecture):
475     architecture = architecture.lower()
476     is_quantized = False
477     if architecture == 'inception_v3':
478         # pylint: disable=line-too-long
479         data_url = 'http://download.tensorflow.org/models/inception_v4_2016_09_09.tar.gz'
480         # pylint: enable=line-too-long
481         bottleneck_tensor_name = 'InceptionV4/Logits/Logits/MatMul:0' #instead of pool_3/_reshape:0
482         bottleneck_tensor_size = 1001 # changed from 2002
483         input_width = 299
484         input_height = 299
485         input_depth = 3
486         resized_input_tensor_name = 'InputImage:0'
487         model_file_name = 'inception_v4.pb'
488         input_mean = 128
489         input_std = 128
490     else:
491         tf.logging.error("Couldn't understand architecture name '%s'", architecture)
492         raise ValueError('Unknown architecture', architecture)
493
494     return {
495         'data_url': data_url,
496         'bottleneck_tensor_name': bottleneck_tensor_name,
497         'bottleneck_tensor_size': bottleneck_tensor_size,
498         'input_width': input_width,
499         'input_height': input_height,
500         'input_depth': input_depth,
501         'resized_input_tensor_name': resized_input_tensor_name,
502         'model_file_name': model_file_name,
503         'input_mean': input_mean,
504         'input_std': input_std,
505         'quantize_layer': is_quantized,
506     }
507
508
509 def add_jpeg_decoding(input_width, input_height, input_depth, input_mean,
510                       input_std):
511     jpeg_data = tf.placeholder(tf.string, name='DecodeJPGInput')
512     decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_depth)
513     decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32)
514     decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
515     resize_shape = tf.stack([input_height, input_width])
516     resize_shape_as_int = tf.cast(resize_shape, dtype=tf.int32)

```

E. PŘETRÉNOVÁNÍ FINÁLNÍ VRSTVY MODELU INCEPTION-V4

```
517     resized_image = tf.image.resize_bilinear(decoded_image_4d,
518                                             resize_shape_as_int)
519     offset_image = tf.subtract(resized_image, input_mean)
520     mul_image = tf.multiply(offset_image, 1.0 / input_std)
521     return jpeg_data, mul_image
522
523
524 def main(_):
525     tf.logging.set_verbosity(tf.logging.INFO)
526     prepare_file_system()
527     model_info = create_model_info(FLAGS.architecture)
528     if not model_info:
529         tf.logging.error('Did not recognize architecture flag')
530     return -1
531
532 # Set up the pre-trained graph.
533 maybe_download_and_extract(model_info['data_url'])
534 graph, bottleneck_tensor, resized_image_tensor = (
535     create_model_graph(model_info))
536
537 # Look at the folder structure, and create lists of all the images.
538 image_lists = create_image_lists(FLAGS.image_dir, FLAGS.testing_percentage,
539                                 FLAGS.validation_percentage)
540 class_count = len(image_lists.keys())
541 if class_count == 0:
542     tf.logging.error('No valid folders of images found at ' + FLAGS.image_dir)
543     return -1
544 if class_count == 1:
545     tf.logging.error('Only one valid folder of images found at ' +
546                       FLAGS.image_dir +
547                       ' - multiple classes are needed for classification .')
548 return -1
549
550 # See if the command-line flags mean we're applying any distortions.
551 do_distort_images = should_distort_images(
552     FLAGS.flip_left_right, FLAGS.random_crop, FLAGS.random_scale,
553     FLAGS.random_brightness)
554
555 with tf.Session(graph=graph) as sess:
556     # Set up the image decoding sub-graph.
557     jpeg_data_tensor, decoded_image_tensor = add_jpeg_decoding(
558         model_info['input_width'], model_info['input_height'],
559         model_info['input_depth'], model_info['input_mean'],
560         model_info['input_std'])
561
562     if do_distort_images:
563         # We will be applying distortions, so setup the operations we'll need.
564         (distorted_jpeg_data_tensor,
565          distorted_image_tensor) = add_input_distortions(
566              FLAGS.flip_left_right, FLAGS.random_crop, FLAGS.random_scale,
567              FLAGS.random_brightness, model_info['input_width'],
568              model_info['input_height'], model_info['input_depth'],
569              model_info['input_mean'], model_info['input_std'])
570     else:
571         # We'll make sure we've calculated the 'bottleneck' image summaries and
572         # cached them on disk.
573         cache_bottlenecks(sess, image_lists, FLAGS.image_dir,
574                           FLAGS.bottleneck_dir, jpeg_data_tensor,
575                           decoded_image_tensor, resized_image_tensor,
576                           bottleneck_tensor, FLAGS.architecture)
577
578     # Add the new layer that we'll be training.
579     (train_step, cross_entropy, bottleneck_input, ground_truth_input,
580      final_tensor) = add_final_training_ops(
581          len(image_lists.keys()), FLAGS.final_tensor_name, bottleneck_tensor,
582          model_info['bottleneck_tensor_size'], model_info['quantize_layer'])
583
584     # Create the operations we need to evaluate the accuracy of our new layer.
585     evaluation_step, prediction = add_evaluation_step(
586         final_tensor, ground_truth_input)
587
588     # Merge all the summaries and write them out to the summaries_dir
589     merged = tf.summary.merge_all()
590     train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train',
591                                         sess.graph)
592
593     validation_writer = tf.summary.FileWriter(
594         FLAGS.summaries_dir + '/validation')
595
```

```

596 # Set up all our weights to their initial default values.
597 init = tf.global_variables_initializer()
598 sess.run(init)
599
600 # Run the training for as many cycles as requested on the command line.
601 for i in range(FLAGS.how_many_training_steps):
602     # Get a batch of input bottleneck values, either calculated fresh every
603     # time with distortions applied, or from the cache stored on disk.
604     if do_distort_images:
605         (train_bottlenecks,
606          train_ground_truth) = get_random_distorted_bottlenecks(
607              sess, image_lists, FLAGS.train_batch_size, 'training',
608              FLAGS.bottleneck_dir, distorted_jpeg_data_tensor,
609              distorted_image_tensor, resized_image_tensor, bottleneck_tensor)
610     else:
611         (train_bottlenecks,
612          train_ground_truth, _) = get_random_cached_bottlenecks(
613              sess, image_lists, FLAGS.train_batch_size, 'training',
614              FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
615              decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
616              FLAGS.architecture)
617
618     # Feed the bottlenecks and ground truth into the graph, and run a training
619     # step. Capture training summaries for TensorBoard with the 'merged' op.
620     train_summary, _ = sess.run(
621         [merged, train_step],
622         feed_dict={bottleneck_input: train_bottlenecks,
623                    ground_truth_input: train_ground_truth})
623     train_writer.add_summary(train_summary, i)
624
625     # Every so often, print out how well the graph is training.
626     is_last_step = (i + 1 == FLAGS.how_many_training_steps)
627     if (i % FLAGS.eval_step_interval) == 0 or is_last_step:
628         train_accuracy, cross_entropy_value = sess.run(
629             [evaluation_step, cross_entropy],
630             feed_dict={bottleneck_input: train_bottlenecks,
631                        ground_truth_input: train_ground_truth})
632         tf.logging.info('%%s: Step %d: Train accuracy = %.1f%%' %
633                         (datetime.now(), i, train_accuracy * 100))
634         tf.logging.info('%%s: Step %d: Cross entropy = %f' %
635                         (datetime.now(), i, cross_entropy_value))
636         validation_bottlenecks, validation_ground_truth, _ = (
637             get_random_cached_bottlenecks(
638                 sess, image_lists, FLAGS.validation_batch_size, 'validation',
639                 FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
640                 decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
641                 FLAGS.architecture))
642
643     # Run a validation step and capture training summaries for TensorBoard
644     # with the 'merged' op.
645     validation_summary, validation_accuracy = sess.run(
646         [merged, evaluation_step],
647         feed_dict={bottleneck_input: validation_bottlenecks,
648                    ground_truth_input: validation_ground_truth})
649     validation_writer.add_summary(validation_summary, i)
650     tf.logging.info('%%s: Step %d: Validation accuracy = %.1f%% (N=%d)' %
651                     (datetime.now(), i, validation_accuracy * 100,
652                      len(validation_bottlenecks)))
652
653     # Store intermediate results
654     intermediate_frequency = FLAGS.intermediate_store_frequency
655
656     if (intermediate_frequency > 0 and (i % intermediate_frequency == 0)
657         and i > 0):
658         intermediate_file_name = (FLAGS.intermediate_output_graphs_dir +
659                                  'intermediate_' + str(i) + '.pb')
660         tf.logging.info('Save intermediate result to : ' +
661                         intermediate_file_name)
662         save_graph_to_file(sess, graph, intermediate_file_name)
663
664     # We've completed all our training, so run a final test evaluation on
665     # some new images we haven't used before.
666     test_bottlenecks, test_ground_truth, test_filenames = (
667         get_random_cached_bottlenecks(
668             sess, image_lists, FLAGS.test_batch_size, 'testing',
669             FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
670             decoded_image_tensor, resized_image_tensor, bottleneck_tensor,
671             FLAGS.architecture))
672     test_accuracy, predictions = sess.run(
673         [evaluation_step, prediction],
674         feed_dict={bottleneck_input: test_bottlenecks,

```

E. PŘETRÉNOVÁNÍ FINÁLNÍ VRSTVY MODELU INCEPTION-V4

```
675         ground_truth_input: test_ground_truth})
676     tf.logging.info('Final test accuracy = %.1f%% (N=%d)' %
677                     (test_accuracy * 100, len(test_bottlenecks)))
678
679     if FLAGS.print_misclassified_test_images:
680       tf.logging.info('==== MISCLASSIFIED TEST IMAGES ====')
681       for i, test_filename in enumerate(test_filenames):
682         if predictions[i] != test_ground_truth[i]:
683           tf.logging.info('%70s %s' %
684                           (test_filename,
685                            list(image_lists.keys())[predictions[i]]))
686
687   # Write out the trained graph and labels with the weights stored as
688   # constants.
689   save_graph_to_file(sess, graph, FLAGS.output_graph)
690   with gfile.FastGFile(FLAGS.output_labels, 'w') as f:
691     f.write('\n'.join(image_lists.keys()) + '\n')
692
693
694   if __name__ == '__main__':
695     parser = argparse.ArgumentParser()
696     parser.add_argument(
697       '--image_dir',
698       type=str,
699       default='',
700       help='Path to folders of labeled images.')
701   )
702   parser.add_argument(
703     '--output_graph',
704     type=str,
705     default='/tmp/output_graph.pb',
706     help='Where to save the trained graph.'
707   )
708   parser.add_argument(
709     '--intermediate_output_graphs_dir',
710     type=str,
711     default='/tmp/intermediate_graph/',
712     help='Where to save the intermediate graphs.'
713   )
714   parser.add_argument(
715     '--intermediate_store_frequency',
716     type=int,
717     default=0,
718     help="""\
719       How many steps to store intermediate graph. If "0" then will not
720       store.\n"""
721     """
722   )
723   parser.add_argument(
724     '--output_labels',
725     type=str,
726     default='/tmp/output_labels.txt',
727     help='Where to save the trained graph\'s labels.'
728   )
729   parser.add_argument(
730     '--summaries_dir',
731     type=str,
732     default='/tmp/retrain_logs',
733     help='Where to save summary logs for TensorBoard.'
734   )
735   parser.add_argument(
736     '--how_many_training_steps',
737     type=int,
738     default=4000,
739     help='How many training steps to run before ending.'
740   )
741   parser.add_argument(
742     '--learning_rate',
743     type=float,
744     default=0.01,
745     help='How large a learning rate to use when training.'
746   )
747   parser.add_argument(
748     '--testing_percentage',
749     type=int,
750     default=10,
751     help='What percentage of images to use as a test set.'
752   )
753   parser.add_argument(
```

```

754     '--validation_percentage',
755     type=int,
756     default=10,
757     help='What percentage of images to use as a validation set .'
758 )
759 parser.add_argument(
760     '--eval_step_interval',
761     type=int,
762     default=10,
763     help='How often to evaluate the training results .'
764 )
765 parser.add_argument(
766     '--train_batch_size',
767     type=int,
768     default=100,
769     help='How many images to train on at a time.'
770 )
771 parser.add_argument(
772     '--test_batch_size',
773     type=int,
774     default=-1,
775     help="""\
776         How many images to test on. This test set is only used once, to evaluate
777         the final accuracy of the model after training completes.
778         A value of -1 causes the entire test set to be used, which leads to more
779         stable results across runs.\n
780         """
781 )
782 parser.add_argument(
783     '--validation_batch_size',
784     type=int,
785     default=100,
786     help="""\
787         How many images to use in an evaluation batch. This validation set is
788         used much more often than the test set, and is an early indicator of how
789         accurate the model is during training.
790         A value of -1 causes the entire validation set to be used, which leads to
791         more stable results across training iterations , but may be slower on large
792         training sets.\n
793         """
794 )
795 parser.add_argument(
796     '--print_misclassified_test_images',
797     default=False,
798     help="""\
799         Whether to print out a list of all misclassified test images.\n
800         """
801     action='store_true'
802 )
803 parser.add_argument(
804     '--model_dir',
805     type=str,
806     default='/tmp/imagenet',
807     help="""\
808         Path to classify_image_graph_def.pb,
809         imagenet_synset_to_human_label_map.txt, and
810         imagenet_2012_challenge_label_map_proto.pbtxt.\n
811         """
812 )
813 parser.add_argument(
814     '--bottleneck_dir',
815     type=str,
816     default='/tmp/bottleneck',
817     help='Path to cache bottleneck layer values as files .'
818 )
819 parser.add_argument(
820     '--final_tensor_name',
821     type=str,
822     default='final_result',
823     help="""\
824         The name of the output classification layer in the retrained graph.\n
825         """
826 )
827 parser.add_argument(
828     '--flip_left_right',
829     default=False,
830     help="""\
831         Whether to randomly flip half of the training images horizontally.\n
832         """

```

E. PŘETRÉNOVÁNÍ FINÁLNÍ VRSTVY MODELU INCEPTION-V4

```
833     action='store_true'  
834 )  
835 parser.add_argument(  
836     '--random_crop',  
837     type=int,  
838     default=0,  
839     help="""\n  
840         A percentage determining how much of a margin to randomly crop off the  
841         training images.\n  
842         """  
843 )  
844 parser.add_argument(  
845     '--random_scale',  
846     type=int,  
847     default=0,  
848     help="""\n  
849         A percentage determining how much to randomly scale up the size of the  
850         training images by.\n  
851         """  
852 )  
853 parser.add_argument(  
854     '--random_brightness',  
855     type=int,  
856     default=0,  
857     help="""\n  
858         A percentage determining how much to randomly multiply the training image  
859         input pixels up or down by.\n  
860         """  
861 )  
862 parser.add_argument(  
863     '--architecture',  
864     type=str,  
865     default='inception_v3',  
866     help="""\n  
867         """)  
868 FLAGS, unparsed = parser.parse_known_args()  
869 tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

Použité programy

TexStudio psaní bakalářské práce v L^AT_EXa překlad do PDF

Notepad++ úprava textových souborů

gedit úprava textových souborů

IntelliJ IDEA Ultimate psaní implementace v jazyce Java

GIT verzování kódu

JDK8 komplikace, ladění a testování Java knihovny

Python 3 zpracování dat, trénování neuronových sítí

Tensorflow softwarová knihovna pro strojové učení

LabelImg označování trénovacích dat

javadoc generování dokumentace do HTML

TexDoclet generování dokumentace do L^AT_EX

wget hromadné stahování dat

bash pomocné skripty pro zautomatizování práce

Obsah přiloženého CD

readme.txt	stručný popis obsahu CD
bin	adresář se spustitelnou formou implementace
doc	adresář s dokumentací implementace
src	
impl	zdrojové kódy implementace
eggdetector	zdrojové kódy implementace knihovny
nn_training	zdrojové kódy pro trénování neuronové sítě
tools	zdrojové kódy pomocných nástrojů k tvorbě bakalářské práce
thesis	zdrojová forma práce ve formátu L ^A T _E X
text	text práce
thesis.pdf	text práce ve formátu PDF