



## ZADÁNÍ BAKALÁ SKÉ PRÁCE

<b>Název:</b>	Detekce objektů v ptáčích hnízdech pomocí neuronových sítí
<b>Student:</b>	Jan Havlík
<b>Vedoucí:</b>	Ing. Josef Pavlásek, Ph.D.
<b>Studijní program:</b>	Informatika
<b>Studijní obor:</b>	Softwarové inženýrství
<b>Katedra:</b>	Katedra softwarového inženýrství
<b>Platnost zadání:</b>	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.

4. 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 4. 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/havluj/EggDetector>>.

---

# 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

<b>Úvod</b>	<b>15</b>
Struktura práce . . . . .	15
<b>1 Cíl práce</b>	<b>17</b>
1.1 Nefunkční požadavky . . . . .	17
1.2 Funkční požadavky . . . . .	17
<b>2 Rešerše</b>	<b>19</b>
2.1 Počítačové vidění . . . . .	19
2.2 Neuronové sítě . . . . .	20
2.3 TensorFlow . . . . .	20
<b>3 Analýza a návrh</b>	<b>23</b>
3.1 Ptáci Online . . . . .	23
3.2 Představení vstupních dat . . . . .	23
3.3 Současný způsob zpracování dat . . . . .	24
3.4 Nefunkční požadavky . . . . .	25
3.5 Funkční požadavky . . . . .	25
3.6 Návrh řešení . . . . .	27
3.7 Volba technologií . . . . .	28
3.8 Shrnutí kapitoly . . . . .	29
<b>4 Nástroje</b>	<b>31</b>
4.1 Hromadné stažení dat . . . . .	31
4.2 Příprava trénovacích a testovacích dat . . . . .	32
<b>5 Implementace detekování objektů</b>	<b>37</b>
5.1 Příprava vývojového prostředí . . . . .	37
5.2 Příprava dat . . . . .	38
5.3 Struktura projektu . . . . .	39
5.4 Trénování neuronové sítě . . . . .	39
5.5 Ověření funkčnosti a výsledky . . . . .	43
5.6 Závěr . . . . .	44

<b>6 Implementace rozpoznávání obrazu</b>	<b>47</b>
6.1 Příprava dat . . . . .	47
6.2 Trénování neuronové sítě . . . . .	49
6.3 Ověření funkčnosti a výsledky . . . . .	50
6.4 Závěr . . . . .	50
<b>7 Implementace Java knihovny</b>	<b>51</b>
7.1 Srovnání implementací neuronových sítí a volba řešení . . . . .	51
7.2 Použití TensorFlow v Javě . . . . .	51
7.3 Uživatelské rozhraní knihovny . . . . .	52
7.4 Implementace . . . . .	52
7.5 Distribuce . . . . .	53
7.6 Výsledek . . . . .	54
<b>8 Ověření implementace</b>	<b>55</b>
8.1 Testování funkčnosti . . . . .	55
8.2 Testování distribuce . . . . .	56
8.3 Měření efektivity implementace . . . . .	56
8.4 Shrnutí kapitoly . . . . .	58
<b>Závěr</b>	<b>59</b>
<b>Literatura</b>	<b>61</b>
<b>A Seznam použitých zkratek</b>	<b>65</b>
<b>B Dokumentace API knihovny</b>	<b>67</b>
B.1 Package org.cvut.havlujal.eggdetector . . . . .	67
<b>C Tagger</b>	<b>73</b>
C.1 Dokumentace (v AJ) . . . . .	73
C.2 Zdrojový kód . . . . .	73
C.3 Ukázkový výstup . . . . .	73
<b>D Folder Trimmer</b>	<b>77</b>
D.1 Dokumentace (v AJ) . . . . .	77
D.2 Zdrojový kód . . . . .	78
<b>E Konfigurace detekování objektů</b>	<b>81</b>
<b>F Test efektivity knihovny</b>	<b>85</b>
<b>G Použité programy</b>	<b>89</b>
<b>H Obsah přiloženého média</b>	<b>91</b>

---

# Seznam obrázků

2.1	Neuron jako základní stavební kámen umělé neuronové sítě. . . . .	20
2.2	Struktura neuronové sítě. . . . .	21
3.1	Ukázka neupraveného snímku z ptačí budky. . . . .	24
3.2	Vejce nemusí být vždy viditelná. . . . .	24
3.3	Program určený ke klasifikaci obrazu. . . . .	27
3.4	Program určený k detekci objektů. . . . .	28
4.1	Tagger – hromadné určování počtu vajec. . . . .	33
4.2	Uživatelské rozhraní nástroje LabelImg. . . . .	35
5.1	Před-trénované modely poskytované společností Google. . . . .	40
5.2	Vizualizace průběhu trénování programem TensorBoard. . . . .	42
5.3	Úvodní obrazovka programu TensorBoard: přehled stavu trénování. . . . .	42
5.4	Použití Google Cloud Platform. . . . .	43
5.5	Model je schopný detektovat všech 9 vajec i přes to, že některá vejce jsou sotva viditelná. . . . .	44
5.6	Model je schopný detektovat i vejce v jiném prostředí. . . . .	45
5.7	Model detekuje vejce i ve složitých podmínkách. . . . .	46
5.8	Méně úspěšný model nedokáže odhalit špatně viditelné vejce. . . . .	46
7.1	Ukázka debuggovacího režimu knihovny EggDetector. . . . .	54



---

# Seznam algoritmů

2.1	Použití algoritmu „stochastic gradient descent“ z knihovny TensorFlow. . . . .	21
3.1	Prvotní návrh uživatelského rozhraní knihovny. . . . .	26
4.1	Hromadné stažení dat ze serveru athena.pef.czu.cz. . . . .	31
4.2	Výběr vhodných složek pro trénování nástrojem Tagger. . . . .	34
	media/labelimg.xml . . . . .	35
7.1	Počítání finálního počtu vajec pro celou složku pomocí dat z jednotlivých snímků.	52
7.2	Transformace vstupního snímku bilineární interpolací. . . . .	53
7.3	Detekce objektů pomocí TensorFlow v Javě. . . . .	53
8.1	Testování softwarové knihovny EggDetector. . . . .	55
	media/imgdata.xml . . . . .	73
D.1	Zdrojový kód nástroje FolderTrimmer . . . . .	78
	media/pipeline.config . . . . .	81
F.1	Test k vyhodnocení efektivity knihovny. . . . .	85



---

# 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 bude vytvořena softwarová knihovna pro určení počtu vajec v hnizdě.



# 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 hnizdě 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 hnizda.

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 JAR<sup>1</sup>.

**F2** Knihovna bude distribuována ve dvou verzích:

- JAR archiv obsahující jak přeložený zdrojový kód knihovny, tak i všechny závislosti<sup>2</sup>, které jsou knihovnou vyžadovány.

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

<sup>2</sup>Zdrojový 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 JPEG<sup>3</sup> a PNG<sup>4</sup>.

---

<sup>3</sup>Joint Photographic Experts Group.

<sup>4</sup>Portable Network Graphics.

# Rešerše

## 2.1 Počítačové vidění

Počítačové vidění je odvětví počítačové vědy, které se zabývá získáváním informací z obrazu. Může se jednat o extrahování informací z videí, obrázků nebo například lékařských snímků [5]. Naučit počítač „vidět“ není vůbec jednoduchý úkol, i přes to, že pro nás mozek je získávání informací z obrazu přirozené a jednoduché. Obraz, jak ho vidí počítač, je pouze posloupnost systematicky uspořádaných jedniček a nul. Motivací pro hledání tvarů nebo přiřazování významů této poslouponosti dat je automatizace úkolů, které typicky musejí vykonávat lidé. Mezi typické využití počítačového vidění patří například [6]:

- autonomní vozidla,
- analýza obrazu v bezpečnostní systémech,
- průmyslové roboty,
- rozpoznávání obrazu.

V této bakalářské práci se budeme zabývat rozpoznáváním obrazu, pomocí kterého se pokusíme automatizovat část lidské práci vykonávané v rámci projektu Ptáci Online. Rozpoznávání obrazu můžeme rozčlenit do několika kategorií podle typu problému [6], například:

**rozpoznávání obrazu:** neboli klasifikace obrazu – zařazení obrazu do jedné ze známých kategorií (např. na snímku je pes)

**detekce objektů:** autonomní vozy se snaží detektovat a následně klasifikovat všechny objekty v jeho okolí

**identifikace:** rozpoznávání obličeje nebo otisku prstu

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

## 2. REŠERŠE

překrývat, mohou být různě barevné, vzdálené nebo otočené. Všechny tyto problémy znemožňují vytvoření perfektního algoritmu manuálně.

V současné době jsou algoritmy umělé inteligence nejefektivnějším řešením rozpoznávání obrazu. Algoritmy založené na konvolučních neuronových sítích se dokonce přiblížily svou přesností a efektivitou lidem[7].

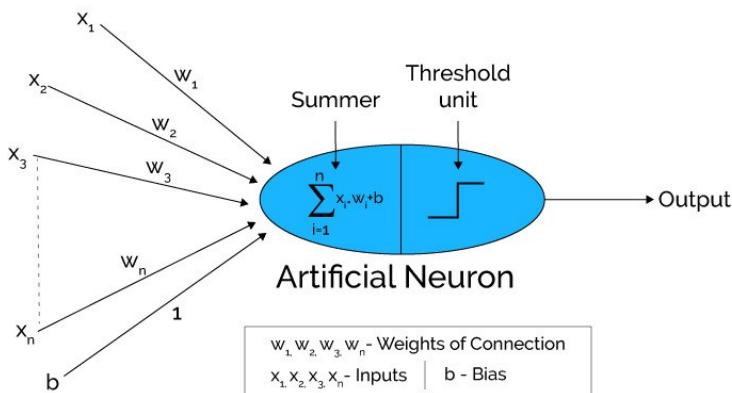
### 2.2 Neuronové sítě

Umělá neuronová síť je softwarový systém inspirovaný biologickými neuronovými sítěmi, které se nachází v lidských s zvířecích mozcích. Neuronové sítě jsou běžně používány například k rozpoznávání obrazu, filtrování spamu nebo předvídání vývoje hodnoty akcií.

Umělé neuronové sítě obsahují neurony, které jsou mezi sebou propojeny. Neuron dostane vstup od předchozích neuronů, tento vstup zpracuje podle své interní logiky a vygeneruje výstup, který je vstupem pro následující neurony. Tako vytvořená neuronová síť je tedy **orientovaným váženým grafem**. Vstupem pro „první řadu“ neuronů jsou vstupní data, například jednotlivé pixely obrazu. Aktivace „poslední řady“ neuronů reprezentuje výsledek pro daný vstup [8]. Struktura neuronové sítě a neuronu je znázorněna na obrázku 2.2, respektive 2.1.

Dnešní architektury neuronových sítí však mohou být mnohem komplexnější. Mohou se lišit počtem skrytých vrstev, strukturou propojení neuronů (například cyklická propojení neuronů), skryté vrstvy mohou mít paměť, atd. Mezi nejznámější typy neuronových sítí patří konvoluční neuronové sítě, modulární neuronové sítě, Boltzmannova síť, síť s dlouhodobou / krátkodobou pamětí a perceptrony. Stručné shrnutí jednotlivých typů viz [9].

V rámci této práce se zaměříme především na použití hlubokých konvolučních neuronových sítí ke klasifikaci obrazu a detekci objektů.



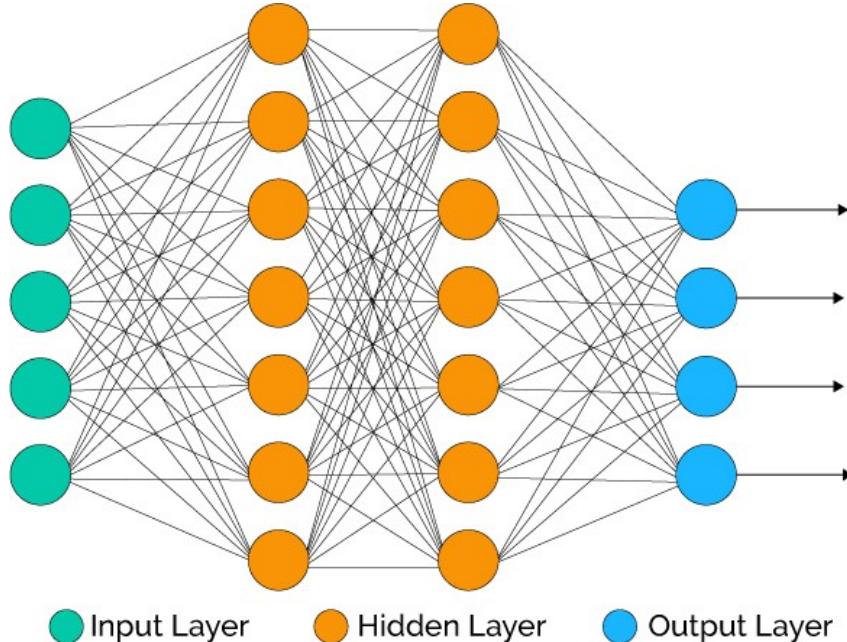
Obrázek 2.1: Neuron jako základní stavební kámen umělé neuronové sítě.

Zdroj: [https://cdn-images-1.medium.com/max/800/0\\*0HlzxsoDSEBXW0iI.jpg](https://cdn-images-1.medium.com/max/800/0*0HlzxsoDSEBXW0iI.jpg) [9].

### 2.3 TensorFlow

„TensorFlow je otevřená softwarová knihovna<sup>5</sup> pro numerické výpočty pomocí metody „data flow graph““ [10]. Knihovnu je možné použít například pro různé matematické výpočty, tvorbu

<sup>5</sup>open source software



Obrázek 2.2: Struktura neuronové sítě.

Zdroj: [https://cdn-images-1.medium.com/max/1600/0\\*IUWJ5oJ\\_z6AiG7Ja.jpg](https://cdn-images-1.medium.com/max/1600/0*IUWJ5oJ_z6AiG7Ja.jpg) [9].

lineárních modelů a neuronových sítí. Použitím knihovny získáme přístup k sadě připravených algoritmů, které nám značně usnadní tvorbu neuronové sítě. K dispozici jsou například algoritmy pro trénování (hledání optimálních vah k jednotlivým hranám grafu), jako je například algoritmus „stochastic gradient descent“. TensorFlow dělá použití dostupných algoritmů opravdu jednoduché, viz algoritmus 2.1.

Algoritmus 2.1: Použití algoritmu „stochastic gradient descent“ z knihovny TensorFlow.

```

1 optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
2 train_op = optimizer.minimize(loss=loss,global_step=tf.train.get_global_step())

```

TensorFlow také značně usnadňuje tvorbu a konfiguraci struktury neuronové sítě, viz [11]. V neposlední řadě můžeme použít nespočet skriptů pro práci s daty [12] nebo TensorFlow Object Detection API [13].



## 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“.“ [14]

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 PoE<sup>6</sup> 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 frekvenci<sup>7</sup>. 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.

---

<sup>6</sup>Power over Ethernet.

<sup>7</sup>Počet 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

Akademičtí 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 [15]. 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 [16].

### 3.6 Návrh řešení

Jádro implementace bude tvořit neuronová síť, která bude vytvořena metodou „učení s učitelem“<sup>8</sup>. 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.

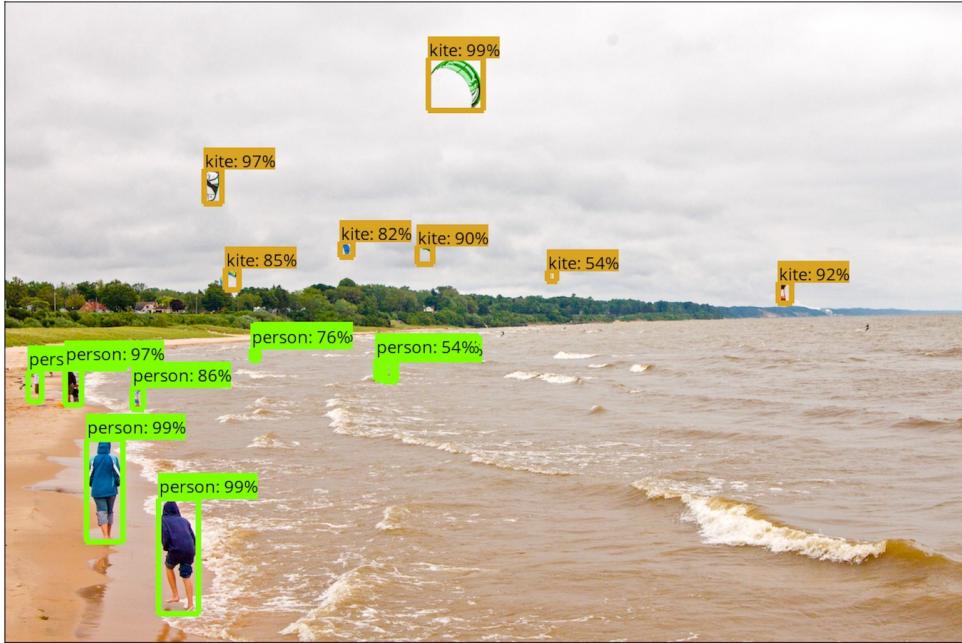
Druhým možným řešením je komplexnější neuronová síť určená k detekci objektů. Obraz je nejprve segmentován na části, o kterých si síť myslí, že by mohly obsahovat nějaké objekty. Následně jsou tyto části klasifikovány a je rozhodnuto, zda-li se jedná o objekt a s jakou pravděpodobností. Výsledkem je 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 3.4. 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

<sup>8</sup>Supervised learning.

### 3. ANALÝZA A NÁVRH

---



Obrázek 3.4: Program určený k detekci objektů.

Zdroj: dokumentace softwarové knihovny TensorFlow [13].

a zůstane barevný – každý bod obsahuje 3 barevné složky<sup>9</sup>), které reprezentuje náš snímek. Knihovna nám poté vrátí požadovaný výsledek<sup>10</sup>.

## 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 FANN<sup>11</sup>. Z těchto tří knihoven pouze TensorFlow poskytuje Java API a už jen z tohoto důvodu je náš nejlepší kandidát.

TensorFlow má skvělou dokumentaci [17] s velkým množstvím kompletních návodů [18]. Poskytuje API k detekci objektů, kde je možné využít již předtrénované modely [13]. Pomocné

<sup>9</sup>RGB - red, green, blue. Každý bod obrázku obsahuje informaci o koncentraci červené, zelené a modré.

<sup>10</sup>V případě rozpoznávání obrazu, knihovna vrátí seznam pravděpodobností pro všechny známe typy. V případě detekce objektů, knihovna vrátí seznam, pozici a typ objektů.

<sup>11</sup>Fast Artificial Neural Network.

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` [19]. Každý způsob implementace vyžaduje mírně odlišný formát trénovacích dat. V kapitolách 5 a 6, které se popisují konkrétní implementace, diskutujeme i přípravu trénovacích dat.

## 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 „pročištění“. 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, použijeme 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 H).

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

```
#!/bin/bash

DIRECTORY=data
URL=http://athena.pef.czu.cz/ptacionline/

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 [20]. 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 [21]

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 [21]. 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 – vejce<sup>12</sup>. 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>
```

<sup>12</sup>Ve 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 repozitóř 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):

## 5. IMPLEMENTACE DETEKOVÁNÍ OBJEKTŮ

---

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

5. Nainstalovat TensorFlow: `sudo pip3 install tensorflow-gpu` nebo `sudo pip3 install tensorflow`. Více informací viz [22].

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 **1890** 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.

Trénovací data přesuneme do pracovní složky. Po přesunutí je nutné opravit XML soubory vygenerované nástrojem LabelImg. V obou složkách, kde máme trénovací data a testovací data pustíme následující příkaz:

```
for fullfile in *.jpg; do
    filename=$(basename "$fullfile")
    filename="${filename%.*}"
    echo "$filename".xml
    awk -v var="$filename" 'NR==3{$0="\t<filename>"var".jpg</filename>"}1;' \
        "$filename".xml > temp.xml && mv temp.xml "$filename".xml
done

for fullfile in *.png; do
    filename=$(basename "$fullfile")
    filename="${filename%.*}"
    echo "$filename".xml
    awk -v var="$filename" 'NR==3{$0="\t<filename>"var".png</filename>"}1;' \
        "$filename".xml > temp.xml && mv temp.xml "$filename".xml
done
```

## 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 H, konkrétně ve složce `src/object-detection-training`.

Všechny skripty, které používáme k trénování a validaci jsou dostupné v příloze H.

## 5.4 Trénování neuronové sítě

Moderní modely pro rozpoznávání obrazu mají miliony parametrů a je **extrémně** výpočetně náročné je vytrénovat. Učení „přenosem modelu“<sup>13</sup> je technika, která ušetří spoustu práce využitím již před-trénovaného modelu a přetrváním pouze finálních vrstev [23]. Trénování sítě „od nuly“ je standardně efektivnější, ale v mé testování jsem ve výsledcích rozdíl nepocítil. Proto jsem se rozhodl o použití již předtrénovaného modelu.

### 5.4.1 Výchozí model

V rámci této implementace jsem zkusil použít tři různé modely, které sloužily jako startovní bod pro trénování. Nejlepší výsledky se dostavily při použití modelu `ssd_mobilenet_v1_coco`<sup>14</sup> [24]. Modely, které nebyly tak úspěšné jsou: `faster_rcnn_nas_coco`<sup>15</sup> a `faster_rcnn_resnet101_coco`<sup>16</sup>. Stažený model umístíme do složky `training` v pracovním adresáři. Další modely, jejich rychlosť a efektivita jsou zobrazeny na obrázku 5.1.

<sup>13</sup>Transfer learning.

<sup>14</sup>Ke stažení na [http://download.tensorflow.org/models/object\\_detection/ssd\\_mobilenet\\_v1\\_coco\\_2017\\_11\\_17.tar.gz](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_2017_11_17.tar.gz) nebo v příloze H.

<sup>15</sup>Ke stažení na [http://download.tensorflow.org/models/object\\_detection/faster\\_rcnn\\_resnet101\\_coco\\_2017\\_11\\_08.tar.gz](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet101_coco_2017_11_08.tar.gz).

<sup>16</sup>Ke stažení na [http://download.tensorflow.org/models/object\\_detection/faster\\_rcnn\\_nas\\_coco\\_2017\\_11\\_08.tar.gz](http://download.tensorflow.org/models/object_detection/faster_rcnn_nas_coco_2017_11_08.tar.gz).

## 5. IMPLEMENTACE DETEKOVÁNÍ OBJEKTŮ

COCO-trained models {#coco-models}

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2 atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes

Obrázek 5.1: Před-trénované modely poskytované společností Google.

### 5.4.2 Konfigurace

Nyní je čas nakonfigurovat parametry, podle kterých se bude neuronová síť řidit při učení. TensorFlow Object Detection API používá soubory `protobuf` ke konfiguraci trénovacího a evaluačního procesu. Konfigurační soubor je rozdělen do 5 částí: `model`, `train_config`, `eval_config`, `train_input_config` a `eval_input_config` [25].

Pro každý typ aplikace jsou vhodné jiné parametry. Výsledný konfigurační soubor použitý společně s modelem `ssd_mobilenet_v1_coco` je vypsán v příloze E.

### 5.4.3 Trénovací data

TensorFlow Object Detection API očekává trénovací data ve formátu `TFRecord` [19]. Do přílohy H jsem přiložil všechny skripty pro vygenerování těchto souborů z trénovacích dat. Nyní máme pouze jednotlivé snímky a k ním XML soubory, které obsahují dodatečné informace o daných snímcích. Cílem je dostat dva soubory: `train.record`, který obsahuje všechna potřebná data pro trénování a `test.record`, který obsahuje všechna data pro validaci a testování. Konverzní skript byl napsán podle předlohy [12]. Abychom dostali dva požadované soubory, budeme postupovat v těchto krocích:

1. Vygenerujeme dva CSV<sup>17</sup> soubory, které budou obsahovat informace o jednotlivých de tekcích v trénovacích a testovacích datech:

```
python3 xml_to_csv.py
```

2. Z kombinace CSV záznamu vygenerovaném v předchozím kroku a trénovacích snímku vytvoříme soubor `train.record`:

```
python3 generate_tfrecord.py --csv_input=data/train_labels.csv \
--output_path=data/train.record
```

<sup>17</sup>Comma Separated Values.

3. Z kombinace CSV záznamu vygenerovaném v prvním kroku a validačních snímků vytvoříme soubor `test.record`:

```
python3 generate_tfrecord.py --csv_input=data/test_labels.csv \
--output_path=data/test.record
```

## 5.4.4 Trénování

V tuto chvíli máme vše připraveno a můžeme začít samotné trénování. Je nutné se uvědomit, že trénování modelu je velmi výpočetně náročné a je možné, že nebudeme mít lokálně dostupný dostatečně výkonný hardware, který by tuto úlohu zvládl v rozumném čase. Máme tedy tři možnosti, kde a jak model vytrénovat. Čas potřebný k trénování je ovlivněn množstvím trénovacích dat, komplexitou modelu a hardwarovým výkonem.

První, nejjednodušší, ale nejméně výkonná varianta je trénování na notebooku nebo stolním počítači pomocí procesoru. Tento způsob je poměrně neefektivní – například při použití metody „transfer learning“ se model trénoval 6 dní čistého času, než se stal použitelným. Trénování takového modelu bez použití metody „transfer learning“ by trvalo týdny.

Druhá varianta, která je komplexnější na zprovoznění je trénování na lokálním stroji pomocí grafické karty. Tato metoda je podstatně více efektivní, než trénování na procesoru počítače. Na dvou desktopových grafických kartách *Nvidia GeForce GTX 650* byl model použitelný po 7 hodinách při použití metody „transfer learning“.

Nejpraktičejší, ale nejsložitější na zprovoznění je trénování na vzdálených serverech. Google poskytuje na platformě *Google Cloud Platform* produkt *ML Engine* [26], který je specializován mimo jiné i na trénování modelů – tzn. je použit optimální hardware. Společnost Google stojí za frameworkm TensorFlow i platformou *ML Engine*, proto je jejich integrace relativně dobře vyřešená. Tento fakt dělá tzv. „cloud learning“ nejfektnivnějším řešením.

### 5.4.4.1 Monitorování průběhu trénování

TensorFlow obsahuje nástroj, kterým můžeme monitorovat průběh trénování modelu. Program spustíme následovně:

```
tensorboard --log_dir=training --port=8080
```

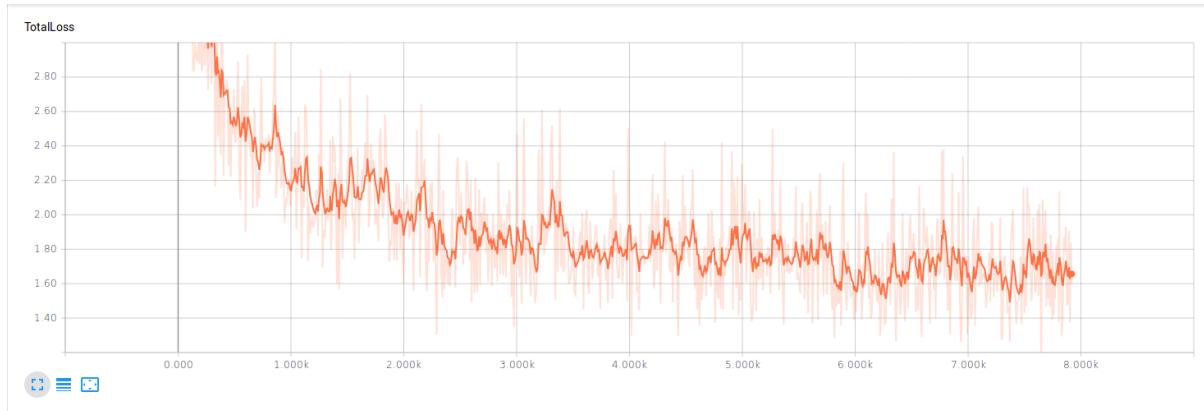
Ve webovém prohlížeči otevřeme adresu `http://localhost:8080` a budeme prezentování úvodní obrazovkou aplikace TensorBoard (viz obr. 5.3). Nástroj vizualizuje veškeré potřebné metriky – jednou z nejzajímavějších je metrika „total loss“ (viz obr. 5.2).

### 5.4.4.2 Trénování modelu na lokálním hardware

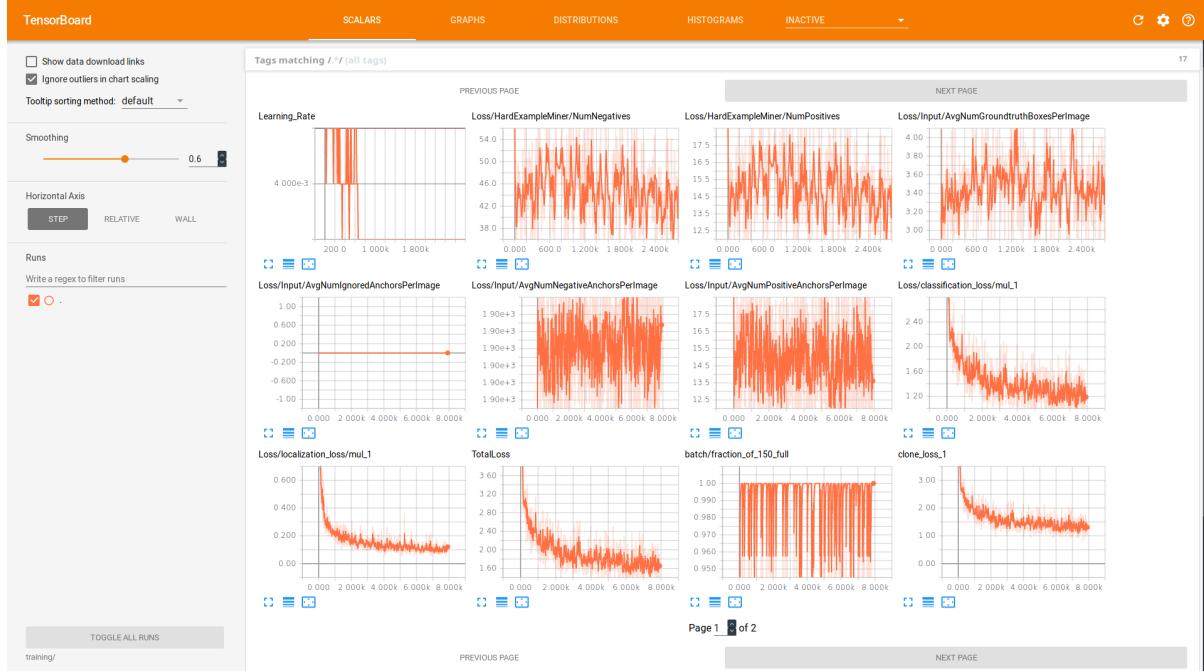
Trénováním na jedné grafické kartě, více grafických kartách nebo na procesoru spouštíme vždy stejným způsobem:

```
python3 $OBJ_DET/train.py --logtosterr --train_dir=training/ \
--pipeline_config_path=training/pipeline.config
```

## 5. IMPLEMENTACE DETEKOVÁNÍ OBJEKTŮ

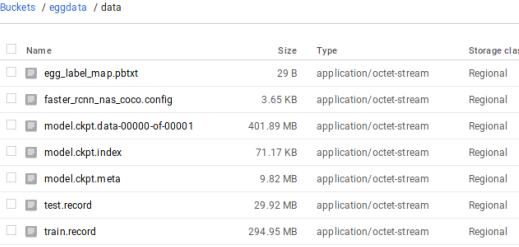


Obrázek 5.2: Vizualizace průběhu trénování programem TensorBoard.

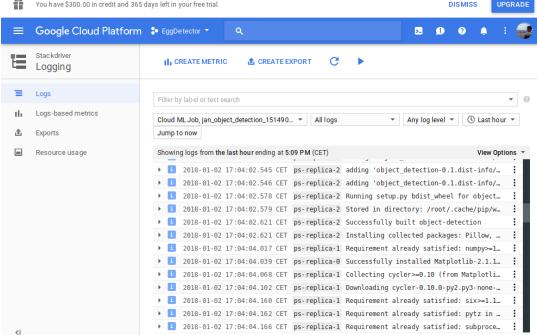


Obrázek 5.3: Úvodní obrazovka programu TensorBoard: přehled stavu trénování.

## 5.5. Ověření funkčnosti a výsledky



Name	Size	Type	Storage class
egg_label_map.pbtxt	29 B	application/octet-stream	Regional
faster_rcnn_nas_coco.config	3.65 KB	application/octet-stream	Regional
model.ckpt.data-00000-of-00001	401.89 MB	application/octet-stream	Regional
model.ckpt.index	71.17 KB	application/octet-stream	Regional
model.ckpt.meta	9.82 MB	application/octet-stream	Regional
test.record	29.92 MB	application/octet-stream	Regional
train.record	294.95 MB	application/octet-stream	Regional



(a) Google Cloud Storage Bucket naplněný daty.

(b) Výstup trénování na Google Cloud Machine Learning Engine.

Obrázek 5.4: Použití Google Cloud Platform.

Parametry můžeme specifikovat, jak a kolik grafických karet má být využito. Například při použití dvou grafických karet jsem specifikoval tyto parametry: `-num_clones=2 -ps_tasks=1`. Více informací viz [27].

### 5.4.4.3 Trénování modelu na Google Cloud ML Engine [26]

Abychom mohli model trénovat na Google Cloud ML Engine, je potřeba několik kroků navíc. Výborný návod, který detailně popisuje, jak zprovoznit trénování na Google platformě viz [28].

Jakmile máme připravené naše uložiště (viz obr. 5.4a), můžeme spustit „vzdálený trénink“:

```
gcloud ml-engine jobs submit training object_detection_${version_unique_ID} \
--job-dir=gs://eggdata/train \
--packages dist/object_detection-0.1.tar.gz,slim/dist/slim-0.1.tar.gz \
--module-name object_detection.train \
--config object_detection/samples/cloud/cloud.yml \
-- \
--train_dir=gs://eggdata/train \
--pipeline_config_path=gs://eggdata/data/faster_rcnn_resnet101_coco.config
```

Průběh trénování je zaznamenáván do „logu“, který je videt na obrázku 5.4b. Samozřejmě můžeme použít i nástroj TensorBoard stejně, jako kdybychom model trénovali lokálně:

```
tensorboard --logdir=gs://eggdata --port=8080
```

## 5.5 Ověření funkčnosti a výsledky

Jakmile se rozhodneme ukončit proces trénování, máme k dispozici několik souborů, které jsou pouhými záhytnými body<sup>18</sup> pro další iterace. Tyto soubory se hodí, kdybychom chteli trénování obnovit tam, kde bylo naposledy ukončeno. Nicméně pro práci s modelem a detekci vajec je potřeba tyto soubory převést na jeden statický. Tomuto procesu se říká „zmražení grafu“. Graf exportujeme následovně:

<sup>18</sup>Checkpoints.

## 5. IMPLEMENTACE DETEKOVÁNÍ OBJEKTŮ

```
python3 export_inference_graph.py \
--input_type image_tensor \
--pipeline_config_path training/ssd_mobilenet_v1_coco.config \
--trained_checkpoint_prefix training/model.ckpt-VERSION \
--output_directory frozen_graph
```

Ve složce `frozen_graph` přibyl soubor typu PB<sup>19</sup>. Tento soubor je výsledným modelem, který můžeme použít k validaci řešení.

K validaci použijeme Python skript napsaný jako Jupyter Notebook [29].

```
jupyter notebook
```

Na přehledové stránce otevřeme soubor `test.ipynb` a zvolíme `Cell → Run All`. Po chvíli jsou prezentovány výstupy skriptu, ve kterém používáme nejúspěšnější model vytištěný metodou „transfer learning“ z modelu `ssd_mobilenet_v1_coco` je vidět na obrázcích 5.5, 5.6 a 5.7. Méně úspěšný model je zobrazen na obrázku 5.8.

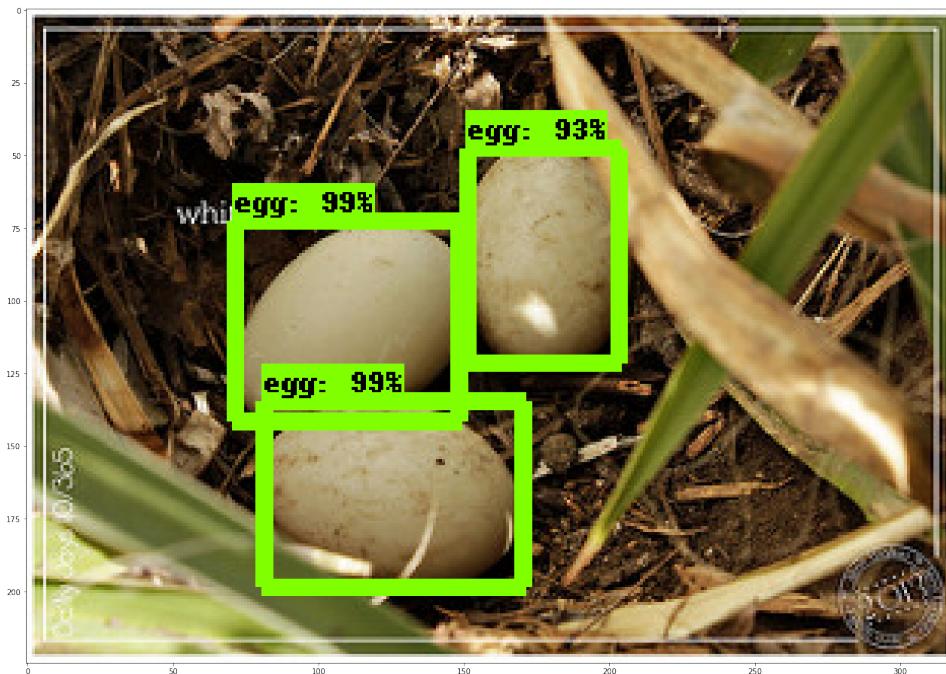


Obrázek 5.5: Model je schopný detektovat všechny 9 vajec i přes to, že některá vejce jsou sotva viditelná.

## 5.6 Závěr

Dokázali jsme, že řešení popsané v této kapitole funguje. Z výsledků vyplývá, že správná konfigurace a výchozí model dokáží značně ovlivnit výsledné chování modelu i přes to, že ho vždy trénujeme stejnými daty.

<sup>19</sup>Protocol Buffer.



Obrázek 5.6: Model je schopný detektovat i vejce v jiném prostředí.

Výsledky doručené modelem, který byl vytrénován metodou transfer learning z modelu `ssd_mobilenet_v1_coco` a používá konfiguraci `training/pipeline.config`, jsou ze všech nejlepší. Proto jsem se rozhodl použít právě tento model jako výsledek této kapitoly a na závěr ho porovnat implementací klasifikace obrazu (viz kapitola 6).

I přes uspokojující výsledky řešení je zde prostor pro vylepšení. Více trénovacích a testovacích dat by zajistilo ještě lepší výsledky (vzorek použitý v této kapitole obsahuje **1890 detekcí**). Předzpracování obrazu tak, aby nikdy nebyl příliš světlý nebo tmavý by zajisté výsledky zlepšilo také. Možná existuje i lepší kombinace konfigurace a výchozího modelu.

Na závěr by bylo vhodné shrnout, že efektivita rozpoznávání vajec touto metodou je **velmi dobrá**.

## 5. IMPLEMENTACE DETEKOVÁNÍ OBJEKTŮ

---



Obrázek 5.7: Model detekuje vejce i ve složitých podmínkách.



Obrázek 5.8: Méně úspěšný model nedokáže odhalit špatně viditelné vejce.

# 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

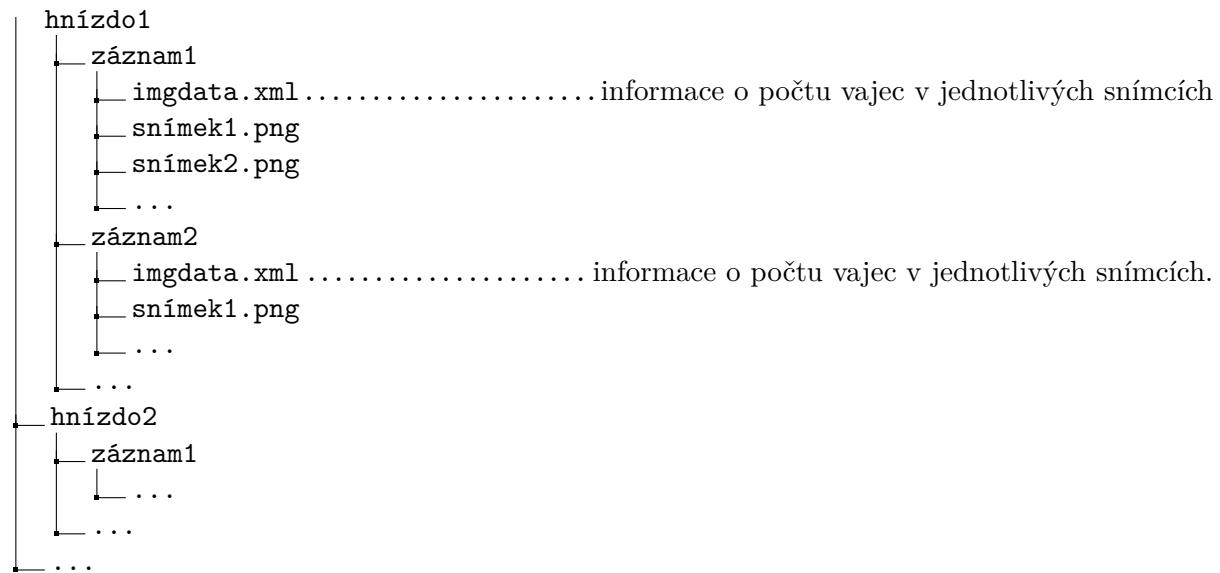
Nástrojem Tagger (viz kapitola C) jsem označil **6178 snímků**. Všechny tyto snímky budou sloužit jako trénovací nebo testovací data pro tuto kapitolu.

## 6. IMPLEMENTACE ROZPOZNÁVÁNÍ OBRAZU

---

Abychom nemuseli psát vlastní skripty pro trénování neuronové sítě, ale mohli použít skripty standartně dostupné [30], 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“<sup>20</sup> je technika, která ušetří spoustu práce využitím již před-trénovaného modelu a přetrénováním pouze finálních vrstev [23]. Více informací k efektivitě tohoto řešení viz kapitola 2.

Předpokladem pro trénování neuronové sítě je nainstalovaná knihovna TensorFlow a všechny její závislosti [22]. Model, ze kterého budeme vycházet je **Inception-v4**<sup>21</sup> [31], který byl vytrénován společností Google na přibližně 1,2 mil. snímků[32]. V soutěži ImageNet [7] drží model Inception-v4 nejlepšího skóre: Top-1 Accuracy<sup>22</sup> 80.2% a Top-5 Accuracy<sup>23</sup> 95.2% [33].

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 H.

### 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
  |  |__ inception_v4.pb ..... nutné pro správné fungování skriptu retrain.py
  |__ training_summaries
  |__ result ..... umístění výsledného modelu
  |__ training_data ..... trénovací data ve formátu specifikovaném výše
```

Poté spustíme připravený skript `retrain.py`:

```
cd egg_recognition
python3 retrain.py \
--bottleneck_dir=bottlenecks \
--how_many_training_steps=8000 \
--model_dir=models/ \
--summaries_dir=training_summaries/ \
--output_graph=result/egg_classifier_graph.pb \
--output_labels=result/egg_classifier_labels.txt \
--image_dir=training_data \
--print_misclassified_test_images
--random_crop=16
--random_scale=7
--random_brightness=4
```

Hodnoty parametrů `how_many_training_steps`, `random_brightness`, `random_scale` a `random_crop` můžeme změnit. K hodnotám uvedeným výše jsem dospěl opakováným testová-

<sup>20</sup>Transfer learning.

<sup>21</sup>Dostupný ke stažení na: [http://download.tensorflow.org/models/inception\\_v4\\_2016\\_09\\_09.tar.gz](http://download.tensorflow.org/models/inception_v4_2016_09_09.tar.gz)

<sup>22</sup>Odpověď modelu (ta s nejvyšší pravděpodobností) přesně odpovídala očekávanému výsledku.

<sup>23</sup>Kterákoli z 5 nejpravděpodobnějších odpovědí modelu odpovídala očekávanému výsledku.

ním a tyto hodnoty produkovaly nejlepší výsledky. Samotný skript `retrain.py` [30] poskytuje nejlepší dokumentaci dostupných parametrů.

### 6.3 Ověření funkčnosti a výsledky

Vytrénovaný a vyexportovaný model můžeme otestovat pomocí veřejně dostupného skriptu `label_image.py` [34]:

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

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

```
5 (score = 0.62071)
4 (score = 0.44595)
6 (score = 0.43252)
0 (score = 0.43049)
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 dominantní<sup>24</sup>.

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.

---

<sup>24</sup>Sít by jedné kategorii přiřadila vysoké skóre ( $> 80\%$ ) a zbytek kategorií by mělo skóre nízké.

# Implementace Java knihovny

Předmětem této kapitoly je tvorba softwarové knihovny, která je cílem této práce. Nejdříve porovnáme výsledky dvou implementací z kapitoly 5 a 6. Jako další krok budeme diskutovat, jakým způsobem integrovat vybranou implementaci do Javy. Následně se zaměříme na uživatelské rozhraní knihovny a implementaci. Na závěr vyřešíme distribuci knihovny nástrojem pro správu, řízení a automatizaci buildů aplikací.

## 7.1 Srovnání implementací neuronových sítí a volba řešení

Vzhledem k výsledkům popsaných v závěru kapitoly 5 a 6 je zřejmé, že detekce objektů popsaná v kapitole 5 je vhodnější implementací.

Nicméně výsledná knihovna je navržena tak, že kdykoliv můžeme styl řešení změnit bez toho, aniž bychom museli měnit uživatelské rozhraní.

## 7.2 Použití TensorFlow v Javě

Abychom mohli použít model pro detekci vajec v naší knihovně, je nutné ho exportovat, stejně jako v kapitole 5.5. Exportovaný model umístíme mezi statické zdroje knihovny.

TensorFlow poskytuje API pro jazyk Java. Abychom ho mohli používat, stačí přidat do pom.xml závislost:

```
1 <dependency>
2   <groupId>org.tensorflow</groupId>
3   <artifactId>tensorflow</artifactId>
4   <version>1.4.0</version>
5 </dependency>
```

Java API, které TensorFlow poskytuje, je poměrně nízko-úrovňové [35]. Abychom si usnadnili práci, použijeme wrapper<sup>25</sup>, který používá TensorFlow pro Android. Stačí jen pár drobných modifikací a můžeme ho použít.

<sup>25</sup>Třída, která „obalí“jinou třídu a přidá jí funkčnost.

### 7.3 Uživatelské rozhraní knihovny

Při tvorbě uživatelského rozhraní vycházíme z prvního návrhu, který vznikl analýzou funkčních požadavků (viz algoritmus 3.1). Kompletní dokumentace API, které knihovna uživateli poskytuje je dostupná v příloze B. Nejlepším zdrojem ukázek použití knihovny jsou testy, které diskutujeme v kapitole 8.

### 7.4 Implementace

Kompletní zdrojové kódy implementace jsou dostupné v příloze H. Knihovnu zkompilujeme nástrojem Maven: `mvn clean package -DskipTests`.

Neuronová síť, která se stará o detekci a klasifikaci objektů na jednotlivých snímcích vrací výsledek ve jako množinu detekcí. Každá detekce se skládá z informací o umístění objektu, typu objektu a pravděpodobnosti v procentech, že se opravdu o daný objekt jedná. Je tedy nutné stanovit hranici<sup>26</sup>, která slouží jako mezník, zda-li je výsledek relevantní, či nikoliv. Jestliže nastavíme hranici například na 30%, knihovna vrátí pouze detekované objekty, u kterých si je jistá alespoň na 30%. Kdybychom nastavili hranici na 0%, knihovna by vrátila všechny možné detekce nalezené v obrázku. Celkový počet objektů na snímku je limitován na 100. V kapitole 8.3 se zaměříme na to, jak nastavená hranice přináší nejlepší výsledky.

Algoritmus 7.1: Počítání finálního počtu vajec pro celou složku pomocí dat z jednotlivých snímků.

```
1 public Integer getFinalCount() {
2     TreeMap<Integer, Integer> scores = new TreeMap<>();
3
4     for (Integer val : imageScores.values()) {
5         if (scores.containsKey(val)) {
6             scores.replace(val, scores.get(val) + 1); // increment
7         } else {
8             scores.put(val, 1);
9         }
10    }
11
12    int bestGuess = 0;
13    while (!scores.isEmpty()) {
14        Map.Entry<Integer, Integer> e = scores.pollLastEntry();
15        if (e.getValue() > 1) { // threshold (how many times do we need the value)
16            return e.getKey();
17        } else if (e.getValue() == 1) {
18            bestGuess = e.getValue();
19        }
20    }
21
22    return bestGuess;
23}
24}
```

---

<sup>26</sup>Threshold.

Algoritmus 7.2: Transformace vstupního snímku bilineární interpolací.

```

1 BufferedImage thumbnail = new BufferedImage(INPUT_SIZE, INPUT_SIZE,
2     BufferedImage.TYPE_INT_RGB);
3 Graphics2D tGraphics2D = thumbnail.createGraphics(); //create a graphics object to
4     paint to
5 tGraphics2D.setBackground(Color.WHITE);
6 tGraphics2D.setPaint(Color.WHITE);
7 tGraphics2D.fillRect(0, 0, INPUT_SIZE, INPUT_SIZE);
8 tGraphics2D.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
9     RenderingHints.VALUE_INTERPOLATION_BILINEAR);
10 tGraphics2D.drawImage(image, 0, 0, INPUT_SIZE, INPUT_SIZE, null);
11
12 // convert img to [INPUT_SIZE, INPUT_SIZE, 3]
13 BufferedImage convertedImg = new BufferedImage(thumbnail.getWidth(),
14     thumbnail.getHeight(), BufferedImage.TYPE_INT_RGB);
15 convertedImg.getGraphics().drawImage(thumbnail, 0, 0, null);

```

Algoritmus 7.3: Detekce objektů pomocí TensorFlow v Javě.

```

1 intValues = ((DataBufferInt) convertedImg.getRaster().getDataBuffer()).getData();
2
3 for (int i = 0; i < intValues.length; ++i) {
4     byteValues[i * 3 + 2] = (byte) (intValues[i] & 0xFF);
5     byteValues[i * 3 + 1] = (byte) (((intValues[i] >> 8) & 0xFF));
6     byteValues[i * 3 + 0] = (byte) (((intValues[i] >> 16) & 0xFF));
7 }
8
9 // Copy the input data into TensorFlow.
10 inferenceInterface.feed(INPUT_NAME, byteValues, 1, INPUT_SIZE, INPUT_SIZE, 3);
11
12 // Run the inference call.
13 inferenceInterface.run(outputNames, logStats);
14
15 // Copy the output Tensor back into the output array.
16 outputLocations = new float[MAX_RESULTS * 4];
17 outputScores = new float[MAX_RESULTS];
18 outputClasses = new float[MAX_RESULTS];
19 outputNumDetections = new float[1];
20 inferenceInterface.fetch(outputNames[0], outputLocations);
21 inferenceInterface.fetch(outputNames[1], outputScores);
22 inferenceInterface.fetch(outputNames[2], outputClasses);
23 inferenceInterface.fetch(outputNames[3], outputNumDetections);

```

## 7.5 Distribuce

Abychom splnili funkční požadavek **F2** (viz kapitola 1.2) o distribuci knihovny, použijeme `maven-assembly-plugin`:

```

1 <plugin>
2   <artifactId>maven-assembly-plugin</artifactId>

```

## 7. IMPLEMENTACE JAVA KNIHOVNY

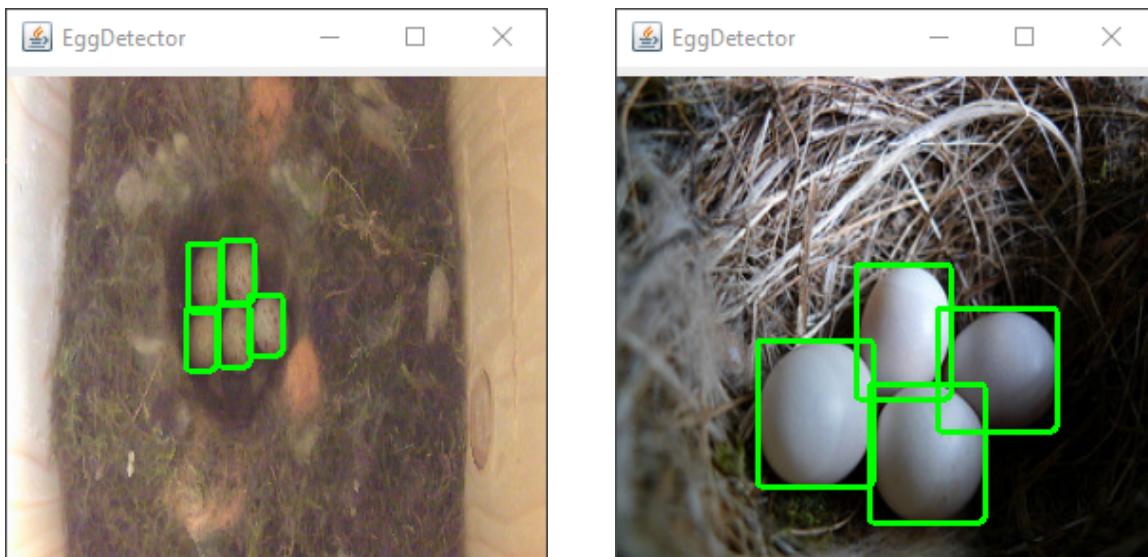
```
3 <configuration>
4   <descriptorRefs>
5     <descriptorRef>jar-with-dependencies</descriptorRef>
6   </descriptorRefs>
7 </configuration>
8 <executions>
9   <execution>
10    <id>make-assembly</id> <!-- this is used for inheritance merges -->
11    <phase>package</phase> <!-- bind to the packaging phase -->
12    <goals>
13      <goal>single</goal>
14    </goals>
15  </execution>
16 </executions>
17 </plugin>
```

## 7.6 Výsledek

Výsledkem této kapitoly je funkční softwarová knihovna v jazyce Java, která splňuje všechny cíle stanovené na začátku práce. Ke knihovně je dostupná kompletní uživatelská dokumentace.

Uživatel získá přístup k detekovanému počtu vajec v požadované složce (výpočet celkového počtu viz alg. 7.1) i k informacím o jednotlivých snímcích. Uživatel má možnost změnit hranici, podle které knihovna určí, jaká podmnožina výsledků je relevantní. Například při nastavení hranice na 50% budou brány v potaz pouze detekce, u kterých si je knihovna jistá alespoň na 50%.

Knihovna obsahuje „debuggovací“ režim, který vizualizuje, jakým způsobem jsou detekce reprezentovány interně. Ukázka „debuggovacího“ režimu je vidět na obrázcích 7.1.



Obrázek 7.1: Ukázka debuggovacího režimu knihovny EggDetector.

# Ověření implementace

V této kapitole se zaměříme testování, validaci a měření efektivity implementace knihovny popsané v kapitole 7. Všechny testy jsou implementovány v jazyce Java pomocí testovacího frameworku JUnit [36]. Zdrojové kódy testů jsou dostupné v příloze H.

## 8.1 Testování funkčnosti

Abych dokázal, že současná implementace knihovny funguje, napsal jsem dva jednotkové testy<sup>27</sup>.

První test využívá pět snímků přiložených ke zdrojovým kódům testů. Nejdříve porovnává předem známý počet vajec v těchto snímcích s počtem, který vypočítá knihovna. Následně je porovnán výsledný počet vajec, který byl vypočítán z těchto 5 snímků.

Algoritmus 8.1: Testování softwarové knihovny EggDetector.

```
1 public class PackagedDataTest {  
2  
3     static EggDetector eggDetector;  
4     static SequenceClassifier sequenceClassifier;  
5  
6     @BeforeClass  
7     public static void setUp() {  
8         eggDetector = new EggDetector();  
9         sequenceClassifier = eggDetector.evaluate(new  
10            File(PackagedDataTest.class.getClassLoader()  
11            .getResource("sample_images").getFile()));  
12    }  
13  
14    @AfterClass  
15    public static void close() {  
16        eggDetector.closeSession();  
17    }  
18  
19    @Test  
20    public void testIndividualScores() {  
21        Map<String, Integer> res = sequenceClassifier.getIndividualCounts();  
22        Assert.assertEquals((int) res.get("image1.png"), 9);  
23    }
```

<sup>27</sup>Unit testing.

## 8. OVĚŘENÍ IMPLEMENTACE

```
22     Assert.assertEquals((int) res.get("image2.png"), 5);
23     Assert.assertEquals((int) res.get("image3.jpg"), 3);
24     Assert.assertEquals((int) res.get("image4.jpg"), 4);
25     Assert.assertEquals((int) res.get("image5.jpg"), 3);
26 }
27
28 @Test
29 public void testFinalScore() {
30     // we expect 3, because we counted 3 eggs 2 times
31     Assert.assertEquals((int) sequenceClassifier.getFinalCount(), 3);
32 }
33 }
```

Druhý test porovnává výsledky knihovny s manuálně označenými daty nástrojem Tagger (viz kapitola C). Vstupem je pouze jedna složka s libovolným počtem snímků.

## 8.2 Testování distribuce

Abychom otestovali, že knihovna splňuje funkční požadavek na distribuci, vytvoříme nový projekt, ve kterém se pokusíme knihovnu použít. Přidáme knihovnu EggDetector do projektu:

```
1 <dependency>
2   <groupId>org.cvut.havlujac1</groupId>
3   <artifactId>eggdetector</artifactId>
4   <version>1.0</version>
5   <scope>system</scope>
6   <systemPath>
7     ${basedir}../../../../bin/eggdetector-1.0-jar-with-dependencies.jar
8   </systemPath>
9 </dependency>
```

a pokusíme se jí použít.

Je vidět, že distribuce funguje. Zdrojové kódy externího projektu se nachází v příloze H ve složce `src/libtest`.

## 8.3 Měření efektivity implementace

Pro určení efektivity naší implementace využijeme manuálně označených **6178 snímků** nástrojem Tagger. Porovnáme také výsledky při různých nastaveních „thresholdu“ knihovny (viz kapitola 7.4).

Zdrojový kód testu je vypsán v příloze F. Knihovnu otestujeme na stejných datech s 5 různými nastaveními „thresholdu“. V nadcházejících sekcích jsou vypsány výsledky.

V testu se zaměříme na dvě metriky. První metrikou je poměr je procentuální úspěšnost knihovny. Tu získáme následujícím způsobem:

$$\text{úspěšnost} = \frac{\text{součet správně určených počtů vajec}}{\text{celkový počet videosekvencí}} * 100$$

Druhou metrikou je rozdíl mezi očekávaným počtem vajec a výsledkem, který dostaneme použitím knihovny:

$$\text{vzdálenost} = \sum_{n=1}^{\text{počet videosekvencí}} |\text{známý počet vajec} - \text{počet vajec vypočítaný knihovnou}|$$

Druhá metrika je lepším nástrojem pro měření efektivity, protože říká, o kolik vajec se knihovna spletla. Z toho vyplívá, že **čím nižší číslo, tím lepší efektivita**.

#### 8.3.1 Test efektivity s nastavením hranice na 60%

```
Found 186 directories.  
EggDetector evaluated 57 directories correctly.  
57/186: 30.645163% success rate.  
Egg count of all folders added together: 982.  
Distance (|real eggs - found eggs|): 409 eggs (smaller is better).
```

#### 8.3.2 Test efektivity s nastavením hranice na 50%

```
Found 186 directories.  
EggDetector evaluated 59 directories correctly.  
59/186: 31.72043% success rate.  
Egg count of all folders added together: 982.  
Distance (|real eggs - found eggs|): 367 eggs (smaller is better).
```

#### 8.3.3 Test efektivity s nastavením hranice na 40%

```
Found 186 directories.  
EggDetector evaluated 62 directories correctly.  
62/186: 33.333336% success rate.  
Egg count of all folders added together: 982.  
Distance (|real eggs - found eggs|): 336 eggs (smaller is better).
```

#### 8.3.4 Test efektivity s nastavením hranice na 30%

```
Found 186 directories.  
EggDetector evaluated 62 directories correctly.  
62/186: 33.333336% success rate.  
Egg count of all folders added together: 982.  
Distance (|real eggs - found eggs|): 311 eggs (smaller is better).
```

#### 8.3.5 Test efektivity s nastavením hranice na 20%

## 8. OVĚŘENÍ IMPLEMENTACE

---

```
Found 186 directories.  
EggDetector evaluated 46 directories correctly.  
46/186: 24.731182% success rate.  
Egg count of all folders added together: 982.  
Distance (|real eggs - found eggs|): 340 eggs (smaller is better).
```

### 8.4 Shrnutí kapitoly

V této kapitole bylo dokázáno, že implementace knihovny funguje a je možné jí použít formou závislostí z jiného projektu. Měřením efektivity knihovny jsme došli k závěru, že nejlepší nastavení hranice pro detekci objektů je **30%**. Knihovnu tedy upravíme tak, aby toto nastavení používala jako výchozí hodnotu.

---

# Závěr

Cílem této práce byl návrh a implementace softwarové knihovny pro určení počtu vajec v ptačích hnízdech. Byla sepsána detailní specifikace funkčních a nefunkčních požadavků, které popisují požadovanou formu knihovny. Prerekvizitou pro implementaci samotné knihovny bylo vyřešení úlohy „rozpoznávání vajec v obraze“.

Čtenáři byla prezentována veškerá potřebná teorie o strojovém učení, neuronových sítích a nástroji TensorFlow. Na základě získaných teoretických znalostí byla provedena analýza funkčních a nefunkčních požadavků; projektu Ptáci Online; současného stavu řešení a struktury vstupních dat. Následně byly navrženy dva směry implementace, kterým bychom mohli úlohu rozpoznávání vajec vyřešit. V poslední části analýzy byly zvoleny vhodné principy a technologie a bylo navrženo uživatelské rozhraní knihovny.

V další kapitole byl čtenář seznámen se všemi nástroji, které jsou potřebné k dokončení implementace. Navrhl a naimplementoval jsem dva plnohodnotné nástroje pro práci s daty a nespočet drobných skriptů pro jejich zpracování. Pomocí těchto nástrojů jsem manuálně připravil **8068** trénovacích dat – označení vajec na snímcích – které byly následně použity k trénování neuronových sítí.

Prvním řešením pro detekci vajec byla implementace realizována technikou „detekce objektů v obraze“. Tato implementace přinesla velmi dobré výsledky a proto jsem se ji rozhodl použít jako základ pro výslednou knihovnu. Druhý styl implementace nebyl tak efektivní a proto jsem se ho rozhodl nepoužít. Nicméně i tento styl implementace je zajímavý a může sloužit jako užitečný základ pro čtenáře, který se bude zabývat rozpoznáváním obrazu.

Nakonec byla implementována samotná knihovna v programovacím jazyce Java, která je schopna automaticky rozpoznat počet vajec v hnizdě. Knihovnu jsem řádně otestoval a změřil její efektivitu.

Do budoucna by bylo vhodné systém rozšířit o zpracování obrazu před samotnou detekcí vajec. Například normalizace jasu, kontrastu a expozice by pomohla výsledky značně zlepšit. Je zde také prostor pro přípravu více trénovacích dat a vylepšení konfigurace neuronové sítě.

Cíl této práce byl v plné míře naplněn, včetně všech funkčních a nefunkčních požadavků. Největším přínosem práce je systém pro detekci objektů v obraze. Vzhledem k návrhu systému stačí, aby byl připraven dostatečně velký vzorek trénovacích dat pro jakýkoliv objekt a můžeme použít tentýž systém například pro detekci mláďat v hnizdech. Tato práce také slouží jako vhodný základ pro čtenáře, kterého zajímá řešení problémů počítačového vidění pomocí neuronových sítí a nástroje TensorFlow.



---

## 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] Ballard, D. H.; Brown, C. M.: *Computer Vision*. Prentice Hall, 1982, ISBN 0-13-165316-4, 523 s., [cit. 2018-01-02]. Dostupné z: [http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/Ballard\\_\\_D.\\_and\\_Brown\\_\\_C.\\_M.\\_1982\\_\\_Computer\\_Vision.pdf](http://homepages.inf.ed.ac.uk/rbf/BOOKS/BANDB/Ballard__D._and_Brown__C._M._1982__Computer_Vision.pdf)
- [6] Klette, R.: *Concise Computer Vision*. Springer, 2014, ISBN 978-1-4471-6320-6, [cit. 2017-10-19].
- [7] Russakovsky, O.; Deng, J.; Su, H.; aj.: ImageNet Large Scale Visual Recognition Challenge. Technická zpráva, Stanford University, University of Michigan, Massachusetts Institute of Technology & UNC Chapel Hill, 2015, [cit. 2017-12-29]. Dostupné z: <https://arxiv.org/pdf/1409.0575.pdf>
- [8] Zell, A.: *Simulation of Neural Networks*. Addison-Wesley, 1994, ISBN 3-89319-554-8, [cit. 2017-11-04].
- [9] XenonStack: *Overview of Artificial Neural Networks and its Applications* [online]. [cit. 2017-11-17]. Dostupné z: [https://www.tensorflow.org/tutorials/image\\_recognition](https://www.tensorflow.org/tutorials/image_recognition)
- [10] Google Inc.: *About TensorFlow* [online]. [cit. 2017-12-06]. Dostupné z: <https://www.tensorflow.org/>
- [11] Google Inc.: *Building the CNN MNIST Classifier* [online]. [cit. 2017-12-06]. Dostupné z: [https://www.tensorflow.org/tutorials/layers#building\\_the\\_cnn\\_mnist\\_classifier](https://www.tensorflow.org/tutorials/layers#building_the_cnn_mnist_classifier)

## LITERATURA

---

- [12] Google Inc.: *TensorFlow: Preparing Inputs* [online]. [cit. 2017-11-17]. Dostupné z: [https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/using\\_your\\_own\\_dataset.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/using_your_own_dataset.md)
- [13] Google Inc.: *Tensorflow Object Detection API* [online]. [cit. 2017-11-17]. Dostupné z: [https://github.com/tensorflow/models/tree/master/research/object\\_detection](https://github.com/tensorflow/models/tree/master/research/object_detection)
- [14] 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](http://www.mzp.cz/cz/news_160608_Ptaci_online)
- [15] 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.
- [16] Google Inc.: *Image Recognition* [online].
- [17] Google Inc.: *Tensorflow API Documentation* [online]. [cit. 2017-11-17]. Dostupné z: [https://www.tensorflow.org/api\\_docs/](https://www.tensorflow.org/api_docs/)
- [18] Google Inc.: *Getting Started with TensorFlow* [online]. [cit. 2017-11-17]. Dostupné z: [https://www.tensorflow.org/get\\_started/](https://www.tensorflow.org/get_started/)
- [19] Google Inc.: *TensorFlow: Importing Data* [online]. [cit. 2017-11-17]. Dostupné z: [https://www.tensorflow.org/programmers\\_guide/datasets](https://www.tensorflow.org/programmers_guide/datasets)
- [20] Pivotal Software, Inc.: *Spring Boot* [online]. 2018. Dostupné z: <https://projects.spring.io/spring-boot/>
- [21] Tzutalin: *LabelImg* [online]. Git code (2015). Dostupné z: <https://github.com/tzutalin/labelImg>
- [22] Google Inc.: *Installing TensorFlow* [online]. [cit. 2017-12-06]. Dostupné z: <https://www.tensorflow.org/install/>
- [23] 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>
- [24] Google Inc.: *Tensorflow detection model zoo* [online]. [cit. 2017-12-17]. Dostupné z: [https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/detection\\_model\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md)
- [25] Google Inc.: *Configuring the Object Detection Training Pipeline* [online]. [cit. 2017-12-01]. Dostupné z: [https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/configuring\\_jobs.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/configuring_jobs.md)
- [26] Google Inc.: *Cloud Machine Learning Engine* [online]. [cit. 2017-10-06]. Dostupné z: <https://cloud.google.com/ml-engine/>
- [27] Chen, W.: *[object detection feature request]: use multiple gpu for training* [online]. [cit. 2017-12-17]. Dostupné z: <https://github.com/tensorflow/models/issues/1972>

- [28] Govindan, S.: *Object Detection: Tensorflow and Google Cloud Platform [online]*. Pub. Nov 5, 2017 [cit. 2017-12-17]. Dostupné z: <https://medium.com/google-cloud/object-detection-tensorflow-and-google-cloud-platform-72e0a3f3bdd6>
- [29] Project Jupyter: *Jupyter Notebook Documentation [online]*. 2017. Dostupné z: <http://jupyter.org/documentation.html>
- [30] 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](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/image_retraining/retrain.py)
- [31] 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>
- [32] Google Inc.: *Preparing the datasets [online]*. [cit. 2017-12-12]. Dostupné z: <https://github.com/tensorflow/models/tree/master/research/slim#preparing-the-datasets>
- [33] 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.
- [34] 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](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/label_image/label_image.py)
- [35] Google Inc.: *org.tensorflow [online]*. [cit. 2017-12-12] (TensorFlow Java API v1.4). Dostupné z: [https://www.tensorflow.org/api\\_docs/java/reference/org/tensorflow/package-summary](https://www.tensorflow.org/api_docs/java/reference/org/tensorflow/package-summary)
- [36] Project Jupyter: *JUnit 5 User Guide [online]*. 2017. Dostupné z: <http://junit.org/junit5/docs/current/user-guide/>
- [37] Oracle: *Javadoc tool. [online]*. [cit. 2017-11-12]. Dostupné z: <http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html>
- [38] *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[37] a TexDoclet[38]. Kód je psán v anglickém jazyce, stejně jako jeho dokumentace. Proto je text i zde v angličtině.

## B.1 Package org.cvut.havlujal.eggdetector

<i>Package Contents</i>	<i>Page</i>
Classes	
EggDetector .....	67
SequenceClassifier .....	71

### 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.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.

- *isDebugMode*

```
1 public boolean isDebugMode()
```

- **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*

## B. DOKUMENTACE API KNIHOVNY

---

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

- **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 médiu (viz příloha H).

### 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 médiu (viz příloha H).

### 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.havlujaj1.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.havlujaj1.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.havlujaj1.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.havlujaj1.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.havlujaj1.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 médiu (viz příloha H).

## 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.sh false ~/eggdetector/data
```

- To delete everything except tagged data, run:

```
target/run.sh true ~/eggdetector/data
```

## D. FOLDER TRIMMER

---

### D.2 Zdrojový kód

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



## Konfigurace detekování objektů

Výsledný konfigurační soubor použitý společně s modelem `ssd_mobilenet_v1_coco`:

```
1 model {
2   ssd {
3     num_classes: 1
4     box_coder {
5       faster_rcnn_box_coder {
6         y_scale: 10.0
7         x_scale: 10.0
8         height_scale: 5.0
9         width_scale: 5.0
10      }
11    }
12    matcher {
13      argmax_matcher {
14        matched_threshold: 0.5
15        unmatched_threshold: 0.5
16        ignore_thresholds: false
17        negatives_lower_than_unmatched: true
18        force_match_for_each_row: true
19      }
20    }
21    similarity_calculator {
22      iou_similarity {
23    }
24  }
25  anchor_generator {
26    ssd_anchor_generator {
27      num_layers: 6
28      min_scale: 0.2
29      max_scale: 0.95
30      aspect_ratios: 1.0
31      aspect_ratios: 2.0
32      aspect_ratios: 0.5
33      aspect_ratios: 3.0
34      aspect_ratios: 0.3333
35    }
36  }
```

## E. KONFIGURACE DETEKOVÁNÍ OBJEKTŮ

---

```
37     image_resizer {
38         fixed_shape_resizer {
39             height: 300
40             width: 300
41         }
42     }
43     box_predictor {
44         convolutional_box_predictor {
45             min_depth: 0
46             max_depth: 0
47             num_layers_before_predictor: 0
48             use_dropout: false
49             dropout_keep_probability: 0.8
50             kernel_size: 1
51             box_code_size: 4
52             apply_sigmoid_to_scores: false
53             conv_hyperparams {
54                 activation: RELU_6,
55                 regularizer {
56                     l2_regularizer {
57                         weight: 0.00004
58                     }
59                 }
60                 initializer {
61                     truncated_normal_initializer {
62                         stddev: 0.03
63                         mean: 0.0
64                     }
65                 }
66                 batch_norm {
67                     train: true,
68                     scale: true,
69                     center: true,
70                     decay: 0.9997,
71                     epsilon: 0.001,
72                 }
73             }
74         }
75     }
76     feature_extractor {
77         type: 'ssd_mobilenet_v1'
78         min_depth: 16
79         depth_multiplier: 1.0
80         conv_hyperparams {
81             activation: RELU_6,
82             regularizer {
83                 l2_regularizer {
84                     weight: 0.00004
85                 }
86             }
87             initializer {
88                 truncated_normal_initializer {
89                     stddev: 0.03
```

---

```
90         mean: 0.0
91     }
92 }
93 batch_norm {
94     train: true,
95     scale: true,
96     center: true,
97     decay: 0.9997,
98     epsilon: 0.001,
99 }
100 }
101 }
102 loss {
103     classification_loss {
104         weighted_sigmoid {
105             anchorwise_output: true
106         }
107     }
108     localization_loss {
109         weighted_smooth_l1 {
110             anchorwise_output: true
111         }
112     }
113     hard_example_miner {
114         num_hard_examples: 3000
115         iou_threshold: 0.99
116         loss_type: CLASSIFICATION
117         max_negatives_per_positive: 3
118         min_negatives_per_image: 0
119     }
120     classification_weight: 1.0
121     localization_weight: 1.0
122 }
123 normalize_loss_by_num_matches: true
124 post_processing {
125     batch_non_max_suppression {
126         score_threshold: 1e-8
127         iou_threshold: 0.6
128         max_detections_per_class: 100
129         max_total_detections: 100
130     }
131     score_converter: SIGMOID
132 }
133 }
134 }
135
136 train_config: {
137     batch_size: 24
138     optimizer {
139         rms_prop_optimizer: {
140             learning_rate: {
141                 exponential_decay_learning_rate {
142                     initial_learning_rate: 0.004
```

## E. KONFIGURACE DETEKOVÁNÍ OBJEKTŮ

---

```
143         decay_steps: 800720
144         decay_factor: 0.95
145     }
146 }
147 momentum_optimizer_value: 0.9
148 decay: 0.9
149 epsilon: 1.0
150 }
151 }
152 fine_tune_checkpoint: "training/ssd_mobilenet_v1_coco_2017_11_17/model.ckpt"
153 from_detection_checkpoint: true
154 num_steps: 200000
155 data_augmentation_options {
156     random_horizontal_flip {
157 }
158 }
159 data_augmentation_options {
160     ssd_random_crop {
161 }
162 }
163 }
164
165 train_input_reader: {
166     tf_record_input_reader {
167         input_path: "data/train.record"
168     }
169     label_map_path: "training/egg_label_map.pbtxt"
170 }
171
172 eval_config: {
173     num_examples: 12
174     max_evals: 10
175 }
176
177 eval_input_reader: {
178     tf_record_input_reader {
179         input_path: "data/test.record"
180     }
181     label_map_path: "training/egg_label_map.pbtxt"
182     shuffle: false
183     num_readers: 1
184     num_epochs: 1
185 }
```

## Test efektivity knihovny

Algoritmus F.1: Test k vyhodnocení efektivity knihovny.

```
1 package org.cvut.havlaja1.eggdetector;
2
3 import java.beans.XMLDecoder;
4 import java.io.File;
5 import java.io.FileInputStream;
6 import java.io.IOException;
7 import java.util.Map;
8
9 import org.cvut.havlaja1.tagger.model.FolderData;
10 import org.cvut.havlaja1.tagger.model.ImgData;
11 import org.junit.AfterClass;
12 import org.junit.BeforeClass;
13 import org.junit.Test;
14
15 public class SuccessRateTest {
16
17     static final String DIR =
18         "/run/media/jan/data/bachelor_thesis_data/data/egg_count_images";
19     static EggDetector eggDetector;
20
21     @BeforeClass
22     public static void setUp() {
23         eggDetector = new EggDetector();
24     }
25
26     @AfterClass
27     public static void close() {
28         eggDetector.closeSession();
29     }
30
31     @Test
32     public void testHighThreshold(){
33         eggDetector.setMinimalConfidence(0.6f);
34         System.out.println(eggDetector);
35         testSuccessRate();
```

## F. TEST EFEKTIVITY KNIHOVNY

---

```
35     }
36
37     @Test
38     public void testMiddleHighThreshold(){
39         eggDetector.setMinimalConfidence(0.5f);
40         System.out.println(eggDetector);
41         testSuccessRate();
42     }
43
44     @Test
45     public void testMiddleThreshold(){
46         eggDetector.setMinimalConfidence(0.4f);
47         System.out.println(eggDetector);
48         testSuccessRate();
49     }
50
51     @Test
52     public void testMiddleLowThreshold(){
53         eggDetector.setMinimalConfidence(0.3f);
54         System.out.println(eggDetector);
55         testSuccessRate();
56     }
57
58     @Test
59     public void testLowThreshold(){
60         eggDetector.setMinimalConfidence(0.2f);
61         System.out.println(eggDetector);
62         testSuccessRate();
63     }
64
65     private void testSuccessRate() {
66         File workingDir = new File(DIR);
67         // get all directories
68         File[] subdirs = workingDir.listFiles((file, name) -> {
69             File tmp = new File(file, name);
70             if (tmp.isDirectory()) {
71                 return true;
72             }
73
74             return false;
75         });
76
77         int totalCnt = 0;
78         int correctCnt = 0;
79
80         // distance
81         int totalEggCount = 0;
82         int lengthDifference = 0;
83
84         for (File dir : subdirs) {
85             try {
86                 int foundTmp = evaluateDir(dir);
87                 int expectedTmp = getExpectedCount(dir);
```

```

88         totalEggCount += expectedTmp;
89         totalCnt++;
90         if (foundTmp == expectedTmp) {
91             correctCnt++;
92         } else {
93             lengthDifference += Math.abs(foundTmp - expectedTmp);
94         }
95         System.out.println("expected: " + expectedTmp + " | found: " +
96                             foundTmp);
97     } catch (IOException | IllegalArgumentException e) {
98     }
99 }
100
101 float cntSuccessRate = ((float) correctCnt) / ((float) totalCnt);
102 System.out.println("Found " + totalCnt + " directories.");
103 System.out.println("EggDetector evaluated " + correctCnt + " directories
104 correctly.");
105 System.out.println(correctCnt + "/" + totalCnt + ": " + (cntSuccessRate * 100)
106 + "% success rate.");
107 System.out.println("Egg count of all folders added together: " + totalEggCount
108 + ".");
109 System.out.println("Distance (|real eggs - found eggs|): " + lengthDifference
110 + " eggs (smaller is better).");
111 }
112
113 private int evaluateDir(File dir) throws IllegalArgumentException {
114     return eggDetector.evaluate(dir).getFinalCount();
115 }
116
117 private int getExpectedCount(File dir) throws IOException {
118     FolderData decodedSettings;
119     try (FileInputStream fis = new FileInputStream(dir.getAbsolutePath() +
120           "/imgdata.xml")) {
121         XMLDecoder decoder = new XMLDecoder(fis);
122         decodedSettings = (FolderData) decoder.readObject();
123         decoder.close();
124         fis.close();
125     } catch (IOException e) {
126         throw e;
127     }
128
129     int eggCount = 0;
130     for (ImgData imgData : decodedSettings.getImgData()) {
131         if (imgData.getEggCount() > eggCount) {
132             eggCount = imgData.getEggCount();
133         }
134     }
135
136     return eggCount;
137 }
138 }
```



## Použité programy

**TexStudio** psaní bakalářské práce v L<sup>A</sup>T<sub>E</sub>Xa 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<sup>A</sup>T<sub>E</sub>X

**wget** hromadné stahování dat

**bash** pomocné skripty pro zautomatizování práce

**awk** hromadná oprava drobných chyb v datech



## Obsah přiloženého média

```
| bin ..... adresář se spustitelnou formou implementace
| doc ..... adresář s dokumentací implementace
| text ..... text práce
|   thesis.pdf ..... text práce ve formátu PDF
|
| src
|   impl ..... zdrojové kódy implementace
|     eggdetector ..... zdrojové kódy implementace knihovny
|     egg_recognition ..... zdrojové kódy implementace rozpoznávání obrazu
|     object-detection-training ..... zdrojové kódy implementace detekce objektů
|     tools ..... zdrojové kódy pomocných nástrojů k tvorbě bakalářské práce
|       foldertrimmer ..... zdrojové kódy nástroje FolderTrimmer
|       tagger ..... zdrojové kódy nástroje Tagger
|       download.sh ..... skript pro hromadné stažení dat
|     libtest ..... projekt pro testování distribuce knihovny
|   thesis ..... zdrojové kód textu práce
| readme.txt ..... stručný popis obsahu média
```