Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Analýza spadu na podložky na dně včelího úlu

Bakalářská práce

Marek Chlubna

Školitel: Ing. Miroslav Skrbek, Ph.D.

České Budějovice 2019

Bibliografické údaje

CHLUBNA, Marek. *Analýza spadu na podložky na dně včelího úlu*. [*Analysis of the fallout on the pads at the bottom of the beehive. Bc. Thesis, in Czech*.] – 36p., České Budějovice, Czech Republic, 2018. Bakalářská práce. Faculty of Science, University of South Bohemia.

Anotace

Bakalářská práce se zabývá metodami strojového vidění a neuronovými sítěmi za účelem detekce a spočítání výskytu kleštíka včelího na podložce vyjmuté ze dna včelího úlu. V práci je provedena rešerše klasických metod strojového vidění vybraných pro detekci objektů a konvolučních neuronových sítí, spolu s jejich implementací. Nad klasickými metodami a neuronovými sítěmi jsou provedeny experimenty, které jsou vyhodnoceny s ohledem na možnou inspiraci výsledky pro vytvoření mobilní aplikace.

Annotation

This bachelor's thesis is dealing with methods of computer vision and use of convolutional neural networks for the purpose of detection and counting the *Varroa destructor* on the pad removed from the bottom of the beehive. A research of such methods and convolutional neural networks is done together with an implementation. Experiments are performed and the results are made with the vision of using the results of this thesis for creating a mobile phone application.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své diplomové práce, a to v nezkrácené podobě, elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích

dne:.....

.....

Marek Chlubna

Poděkování

Rád bych toto srdečné poděkování věnoval mému školiteli panu Ing. Miroslavu Skrbkovi, Ph.D. a panu Ing. Václavu Krištůfkovi, CSc., za jejich odbornost a trpělivost při vzniku práce, cenné rady, čas strávený při konzultacích a na včelnicích. Bez nich by nebylo možné aby tato práce vznikla. V neposlední řadě bych rád poděkoval své rodině, přátelům a dobré kávě za podporu během studia.

Práce byla financně podpořena projektem Strategie AV 21, ROZE, Zdraví včel, 2018.

Obsah

1 Úvod

Tempo rozvoje techniky a inteligentních zařízení, za několik posledních dekád, narostlo takovým způsobem, že se v dnešní době nachází použití výpočetních technologií, v tomto případě aplikací zpracovávajících obraz, napříč širokou škálou různých (nejen vědeckých) odvětvích. Uplatňují se v dopravě, v medicíně, při automatizovaném vyhodnocování lékařských snímků, v geografii a velkou oblast využití nachází i v biologii. Konkrétně i v takových, po technické stránce okrajových oblastech, jako je například včelařství. Jsou hnacím motorem ve zdokonalování práce s nimi a prostředkem k zautomatizování činností.

Práce vznikala ve volné návaznosti na projekt vytvoření inteligentního včelího úlu, který je realizován v rámci projektu Strategie AV 21, Rozmanitost života a zachování ekosystémů (ROZE) na Biologickém centru AV ČR, v. v. i. v Č. Budějovicích. Projekt řeší technickou konstrukci včelího úlu pro sběr a následnou analýzu dat pomocí senzorů, pro snímání vlhkosti, teploty, hmotnosti úlu, zvukových záznamů včel, a vytvoření potřebného softwaru pro zpracování a ukládání dat.

Tato práce se podle provedené rešerše jako první zabývá tématem detekce a spočítáním kleštíka včelího na spadové podložce pomocí kombinace klasických metod strojového vidění a neuronových sítí. Pomocí klasických metod již byla na trhu uvedena aplikace podobného rázu, avšak před každým pořízením snímku je nutné nastavit klíčové parametry, čili postrádá generalizaci, a i tak její úspěšnost není zdaleka optimální. Experimenty v těchto oblastech popsané v práci, mimo jiné, ukazují, že za pomocí vhodné kombinace algoritmů pro zpracování obrazu a konvolučních neuronových sítí lze roztoče detekovat a spočítat bez nutnosti velkého výpočetního výkonu, či velkého časového intervalu vyhodnocování.

1.1 Motivace

Kleštík včelí *(Varroa destructor)* je v současnosti nejvážnější onemocnění včel. Jeho včasná detekce, kterou se tato práce zabývá, je pro léčení včelstev a zabránění jejich vymírání klíčovým tématem. Znalost počtu jedinců kleštíka včelího ve včelstvu umožňuje včelaři nalézt správnou zootechniku k jeho potlačení.

Včelařství v naší společnosti sehrává důležitou roli při opylování rostlin a zemědělských plodin, které tvoří více než 30% globální produkce potravin, udržování rozmanitosti naší přírody a má pro nás i kulturní význam [1].

1.2 Cíl práce

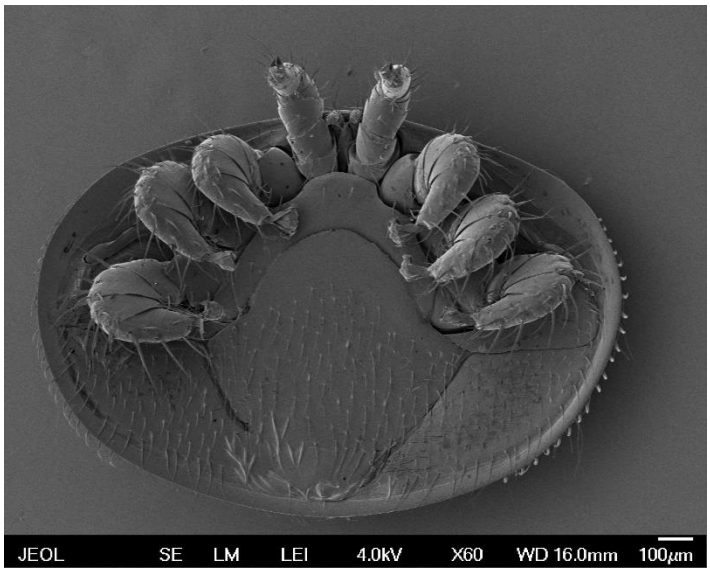
Cílem práce je prozkoumat metody, založené na klasických metodách i neuronových sítích, které by umožnily strojové zpracování a spolehlivé vyhodnocení snímku spadu na podložce v úlu. Takové vyhodnocení by mělo vést k detekci počtu roztočů. V případě úspěšné implementace je možné se inspirovat výsledky práce při tvorbě mobilní aplikace, určenou pro analýzu spadu, která by včelařům ulehčila a zrychlila rozhodování o léčení včelstev. Pro naplnění hlavního cíle práce je nezbytné splnit několik dílčích úkolů:

- Provést rešerši daných metod
- Navrhnout řešení založené na základě rešerše
- Provést experimenty nad navrženými řešeními
- Vyhodnotit experimenty s ohledem na přesnost a výpočetní čas

2 Analýza problému

Roztoč kleštík včelí byl do Evropy zavlečen z Asie a včela medonosná (*Apis mellifera L.*) se mu nedokáže dostatečně bránit [2].

Samičky roztoče jsou díky jejich specifickému eplipsovému tvaru, velikosti (1,1 – 1,5 mm dlouhé a široké 1,5 – 1,9 mm) a barvě (červenohnědé až hnědě lesklé zbarvení) na spadové podložce rozpoznatelné pouhým okem, uvnitř úlu však povrch těla roztoče téměř dokonale splývá s povrchem včelího těla. Hřbet roztoče, jak je vidět na obázku 1, je pokrytý tvrdým štítem, který kryje celé tělo spolu se čtyřmi páry končetin a ústním ústrojím.

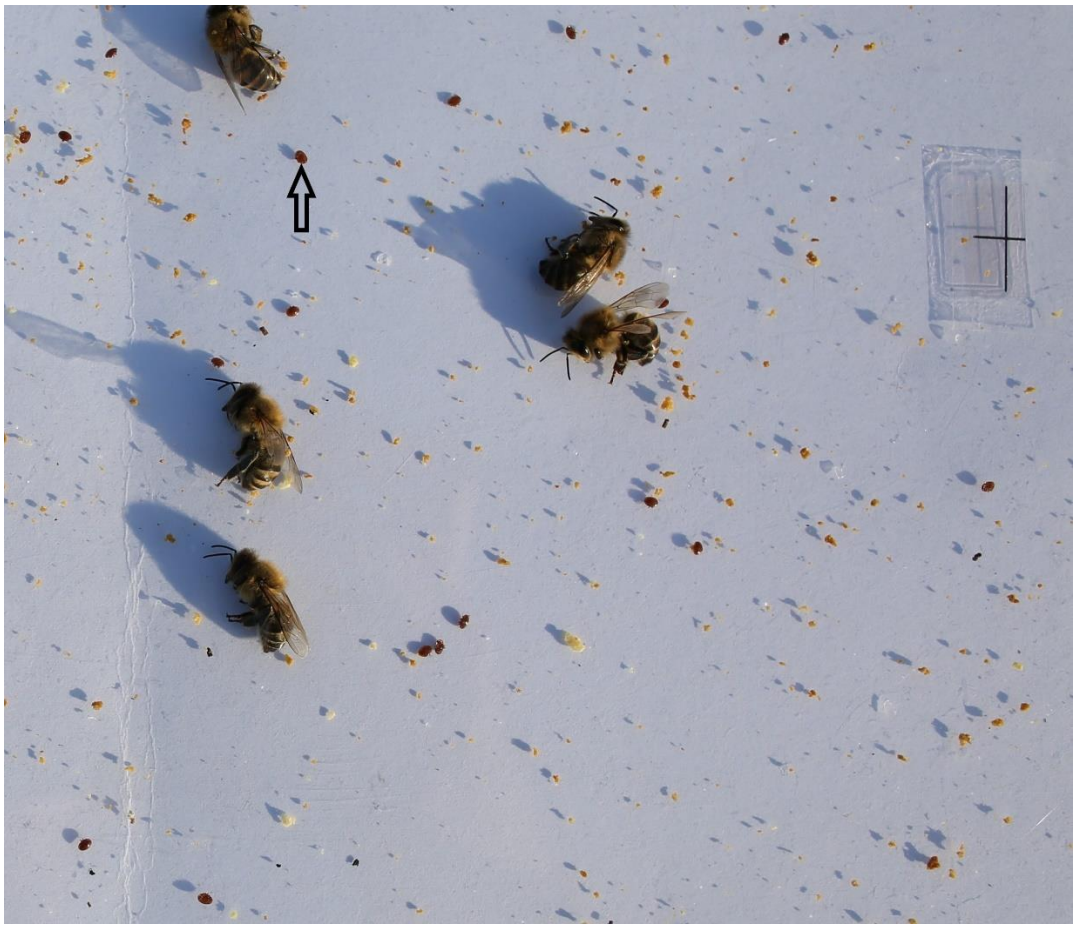


Obrázek 1: Kleštík včelí (*Varroa destructor*). Skenovací elektronová mikroskopie, měřítko 100 µm (Foto V. Krištůfek).

Roztoč se vyvíjí na včelím plodu, kde před jeho zavíčkováním pronikne samička do buňky a s odstupem pár dní do ní naklade vajíčka. První se vylíhne sameček a poté samičky, které sameček následně oplodní. Samečci po spáření uhynou a samičky se přichytí na dospělou včelu a spolu s ní opustí buňky. Ve včelstvu jsou nejvíce nakaženi trubci, kteří roztoče přelétáváním zanáší i do jiných včelstvech. Více nežli samotná přítomnost většího množství samiček roztoče ve včelstvu, jsou problémem virová onemocnění, které kleštík na včely přenáší. S větším počtem roztočů šance virových onemocnění stoupá [2].

Včasná diagnostika kleštíka včelího je klíčová pro zamezení zvýšení počtu roztočů ve včelstvu. Roztoč je kvůli stísněnému prostoru ve včelím úlu těžko technicky

detekovatelný, ale včely se ho snaží ze svého těla svépomocí strhávat a kleštík padá na spadovou podložku. Jejím vyjmutím z úlu (obrázek 2), můžeme, aplikací vhodných metod, spočítat množství jeho výskytu, které je pro včelaře důležitým měřítkem při dalších postupech léčení.



Obrázek 2: Spad kleštíka včelího na podložce (pro příklad je jeden expemplář označen šipkou).

3 Zpracování obrazu

Dnešní doba zaznamenala výrazný pokrok při zpracování obrazů pomocí počítačových nástrojů. Algoritmy zpracovávající obraz jsou dnes již běžnou součástí našeho života, ať je to analýza fotografií na sociálních sítích, za účelem získání informací o uživateli a následnému aplikování cílené reklamy. Vědecké a výzkumné oblasti analyzují fotografie ve snaze objevení nových hvězd, či vesmírných útvarů. V lékařství jsou rentgenové / ultrazvukové snímky zkoumány za lepší diagnostikou pacienta a hledáním abnormálních útvarů v lidském těle. Stále se však vyskytují oblasti, kde má lidské vnímání navrch oproti tomu strojovému, ale právě v těchto případech se vývojáři systémů snaží o dosažení výsledků, díky kterým by mohlo být lidské vnímání posunuto kupředu a předčilo tak naše biologické předpoklady.

Zpracování obrazu pro rozpoznávání objektů se obecně skládá ze čtyř následujících funkčních bloků, které lze metodicky řešit i samostatně [3]:

- Snímání obrazu
- Předzpracování a segmentace obrazu
- Popis objektů a jejich vlastností
- Klasifikace objektů

Následující podkapitoly popíší některé principy těchto funkčních bloků a metody, které byly pro tuto práci použity. Snímání obrazu je z velké části předurčeno použitou technikou, která je rozebrána v následující podkapitole. Popis objektů a jejich klasifikaci v této práci řeší neuronové sítě, z těchto důvodů zde nebudou obsaženy.

3.1 Pořizovací technika

Bylo pořízeno 75 snímků z podložek úlů několika včelnic a dalších 265 snímků bylo nasimulováno pomocí náhodného rozmístění roztočů, měly a dalšího spadového odpadu. To bylo provedeno na papírové, plastové, dřevěné podložky a snímáno při různých vzdálenostech či s mírnou destabilizací obrazu pořizovacího zařízení.

Pro přiblížení se širokému spektru koncových uživatelů byly snímky pořízeny třemi různými zařízeními, od mobilních telefonů po digitální fotoaparáty. 100 snímků bylo pořízeno digitálním fotoaparátem Nikon D6100 s rozlišením 4633*3633 (obrázek 1 je ořezaným snímkem pořízeným tímto fotoaparátem). 80 snímků bylo

pořízeno mobilním telefonem iPhone 5s 2400*1260 a 60 snímků bylo pořízeno za užití mobilního telefonu Lenovo A536 se stejným rozlišením, jako u předchozího zařízení.

3.2 Předzpracování obrazu

Metody předzpracování obrazu obecně slouží k usnadnění následného zpracování obrazu. Cílem těchto metod je potlačit šum, který vznikl při vytváření obrazu, odstranit zkreslení poděděné vlastnostmi snímacího zařízení a snímaných objektů, zvýraznění nebo potlačení specifických rysů objektů [3].

Mezi nejdůležitější metody předzpracování, a zároveň mezi metody využité v této práci, patří:

a) Převod na stupně šedi.

b) Změna barevného modelu (HSV)

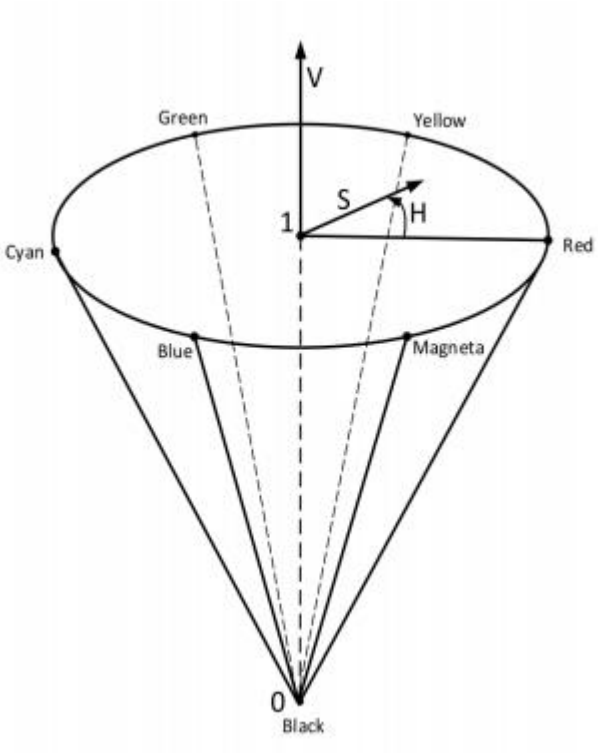
c) Redukce šumu, zaostření obrazu

d) Filtrace

3.2.1 Barevný model HSV

Barevný model HSV, někdy označován jako HSB, je v porovnání s jinými barevnými modely, jako je RGB a CMYK, našemu chápání barev nejbližší [4, 19]. Skládá se také ze tří složek, avšak na rozdíl od ostatních modelů je v něm informace o barvě obsažena v první složce nazývané odstín (H, *hue*). Konkrétní odstíny barev jsou zde obecně vyjádřeny úhlem od 0 do 360 stupňů (při použití knihovny OpenCV to je 180), úhel pak reprezentuje převládající spektrální barvu. Druhou složkou je sytost (S, *saturation*), která je určována v rozmezí hodnot od 0 do 1 a značí čistotu dané barvy. Třetí a poslední složkou je hodnota jasu (V, *value*), pohybuje se mezi hodnotami 0 až 1 a udává množství bílého světla. Samotné grafické vyjádření modelu je ilustrováno na obrázku 3.

Nespornou výhodou při práci s tímto modelem je možnost oddělení jasu a sytosti a práce s nimi bez ohledu na barevný tón. Při použití knihovny OpenCV pro práci s tímto modelem je užíváno k převodu mezi soustavami RGB (OpenCV využívá BGR) a HSV příkazu *BGR2HSV* respektive *HSV2BGR*.



Obrázek 3: Barevný model HSV. Zdroj: [19]

3.2.2 Redukce šumu v obraze

Šumem rozumíme pravidelné či náhodné narušení obrazové funkce $f(i, j) = x_{ij}$, popisující ideální reprezentace zachycené scény [5]. Snímáním fotoaparátem obvykle vzniká barevný náhodný šum, způsobený CCD/CMOS snímačem použitého zařízení. Tento efekt je často umocněný pomocí použité komprese obrazu (například formát .jpeg). Za účelem zmírnění této degradace obrazu, při základním zpracování, jsou ve většině případů použity lineární filtry (Gaussovo vyhlazení) a rank-order filtry (mediánový filtr), které byly využívány i v této práci.

Gaussovo vyhlazení vychází z Gaussovy funkce vyjádřené ve dvourozměrném prostoru jako:

$$\mathrm{G}(\mathrm{x},\mathrm{y}) = \frac{1}{2\pi\sigma 2}\, e^{\frac{x^2+y^2}{2\sigma 2}} \tag{1}$$

Tato funkce je diskretizována na podobu konvolučního jádra (nejčastěji rozměry 3x3, 5x5, či 7x7) a aplikována na obrazovou funkci. Jejím výsledkem je rozostřený obraz, který je zbaven šumu kamery. Občasným použitím však částečně přicházíme o informace obsažené v obrazu, jako je především snížená ostrost obrazu na hranách objektů [5].

Mediánový filtr obrazovou funkci vyhlazuje za použití vypočtené hodnoty mediánu ve stanoveném okolí (často 3x3, 5x5), výsledek je poté přiřazen centrálnímu bodu okolí. Oproti Gaussovému vyhlazení je vhodnější k odstranění náhodného šumu, jelikož zachovává významné hrany a odstraňuje pouze drobné lokální změny v obrazové funkci [5].

3.3 Segmentace obrazu

Jedním z nejtěžších kroků zpracování obrazu je segmentace obrazu [6]. Jedná se o analýzu vedoucí k rozdělení obrazu do částí odpovídajících objektům, které jsou bodem zájmu v dalším průběhu zpracování. Výsledkem segementace by měl tedy být soubor oblastí, které odpovídají objektům ve vstupním obraze. Jedná se pak o tzv. kompletní segmentaci, která využívá vyšší úrovně zpracování, založené na znalostech řešeného problému a není vhodná pro obecné řešení napříč širokou škálou různorodých obrazů [6]. V této práci je pro obecnost využívaná částečná segmentace, nazývaná také segmentace prahováním, která je založena na pricipu homogenity obrazových vlastností (např. barva, jas) uvnitř segmentu.

Prahování je transformace vstupního obrazu na výstupní, často binární obraz, kde je objektům, které mají větší hodnotu nežli práh, přiřazena 1, ostatní objekty (pozadí) mají přiřazenou hodnotu 0. Takto popsané prahování se nazývá prosté.

3.4 Knihovna OpenCV

OpenCV je zkratkou pro *Open Source Computer Vision Library*. Je nejvíce rozšířenou knihovnou, která obsahuje algoritmy pro zpracování obrazu a disponuje také velmi propracovanou dokumentací. Knihovna má modulární strukturu, jedná se o to, že se jednotlivé moduly věnují různým problematikám zpracování obrazu. Nejpoužívanějšími moduly jsou ty, které slouží pro zpracování fotografií, videa a pro generování výstupů v podobě jednoduchých grafických uživatelských rozhraní.

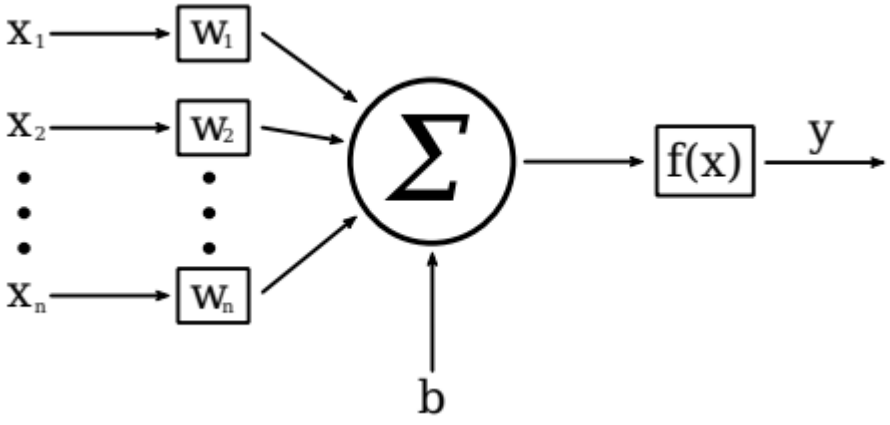
Knihovna lze přeložit na téměř všech operačních systémech a je dostupná pod BSD licencí, to znamená, že je volně použitelná ke komerčním i akademickým potřebám [7]. K práci s touto knihovnou lze využít vícero programovacích jazyků, v případě této práce se bude jednat o jazyk Python.

4 Neuronové sítě

Biologické struktury neuronů u živých organismů inspirovaly vznik umělých neuronových sítí. Tyto počítačové modely nachází uplatnění v úlohách z oblasti počítačového vidění, strojového překladu, rozpoznávání řeči, či různých druhů klasifikace. Základním prvkem je zde také **neuron**, jedná se však o matematický model neuronu biologického (obrázek 4).

Do umělého neuronu je napojen určitý počet **vstupů**. Jednotlivé vstupy jsou obecně reprezentovány čísly z množiny reálných čísel. Počet vstupů lze chápat jako vstupní vektor $x = (x1, x2, x3, ..., xn)$. Každý vstup je v první řadě vynásoben odpovídající **váhou**, ty lze taktéž chápat jako vektor $w = (w1, w2, w3, ..., wn)$. Tyto vstupní hodnoty jsou dále sečteny a v případě existence skalární veličiny v neuronu tzv. **biasu** je k tomuto součtu vstupů přičten ještě bias. Dále je v neuronu obsažena **přechodová funkce**, jejíž výstupem je výstup celého neuronu. Ta je definována vztahem 2 [8].

$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{2}$$



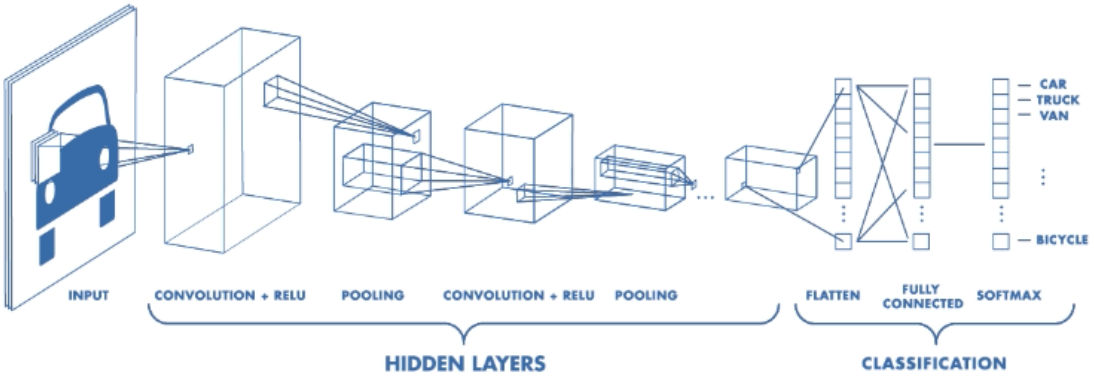
Obrázek 4: Matematický model neuronu.

Neurony jsou spojeny do **vrstev**, které dalším propojováním, na základě zvolené architektury, vytváří neuronovou síť. V této prací jsou použity sítě učené pomocí tzv. *supervised learningu*, tedy **učení s učitelem**, kdy jsou síti předkládány vzory z **trénovací množiny**, reprezentující data, která se má síť naučit zpracovat. Očekávaný

výsledek je pak opakovaně porovnáván s odezvou sítě a váhy spolu s biasem jsou na základě tohoto porovnání upravovány tak, aby se výstup sítě očekávanému výsledku přiblížil [9]. Jednu tuto iteraci nazýváme **epochou**.

4.1 Konvoluční sítě a jejich parametry

Konvoluční sítě jsou složeny ze zřetězených individuálních bloků, které plní rozdílné úlohy. Tyto stavební bloky jsou nejčastěji označováni jako vrstvy konvoluční neuronové sítě.



Obrázek 5: Ilustrační schéma konvoluční sítě[1].

Jak je možné pozorovat na obrázku 5, neuronové sítě mohou mít obecně libovolný počet skrytých vrstev. Sítě můžeme při větším počtu těchto vrstev označit za **hluboké**. Není však udána přesná hranice, která by rozlišila hluboké učení od mělkého. Hovoříme-li tedy o hlubokých sítích, rozumíme ty, které vykazují větší míru složených vrstev nežli je u klasických metod strojového učení obvyklé [9].

4.1.1 Konvoluční vrstva

Konvoluční vrstva je využívána k extrakci příznaků (hrany, rohy, změna barvy atd.) (z anglického *features*) z obrazu [10]. Tvoří ji filtry, čtvercové matice, s nastavenými váhami. Pomocí těchto filtrů se vypočítá konvoluce nad celým obrazem a výsledkem je mapa příznaků. Počet těchto map je roven počtu filtrů, které se v konvoluční vrstvě použijí. Při výpočtu konvoluce se filtr může po obraze pohybovat krokem o určité velikosti, nejčastějším použitím je velikost 1, která zachovává velikost vstupu.

[1] Zdroj: https://cdn-images-1.medium.com/max/800/1*NQQiyYqJJj4PSYAeWvxutg.png

V průběhu definování vrstvy se též určuje velikost použitého filtru, typicky se volí rozměry pomocí lichých čísel jako 3x3, 5x5 či 7x7.

4.1.2 Nelineární vrstva

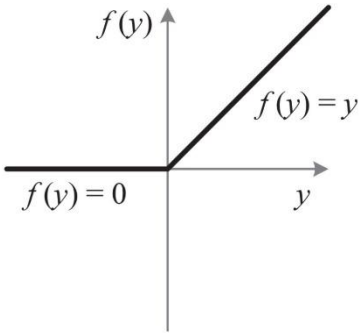
Nelineární vrstva (*Non-Linearity layer*) udává na výstupu aktivační mapu, která byla spočítána aplikováním aktivační funkce na každý prvek vstupu. Nejoblíbenější funkcí je v konvolučních sítích usměrněná lineární funkce (*rectified linear unit*, **ReLU**), definovaná vztahem 3[10].

$$f(\xi) = max(0, \xi) \tag{3}$$

Z rovnice vyplývá, že záporné hodnoty příznakové mapy jsou nahrazeny nulou.
Pro derivaci ReLU platí:

$$f\,0\,(\xi) = \begin{cases} 1\, pro\, \xi > \theta, \\ 0\, pro\, \xi \leq \theta, \end{cases} \tag{4}$$

kde θ je práh, jehož hodnota je obvykle 0.



Obrázek 6: Aktivační funkce ReLU[2].

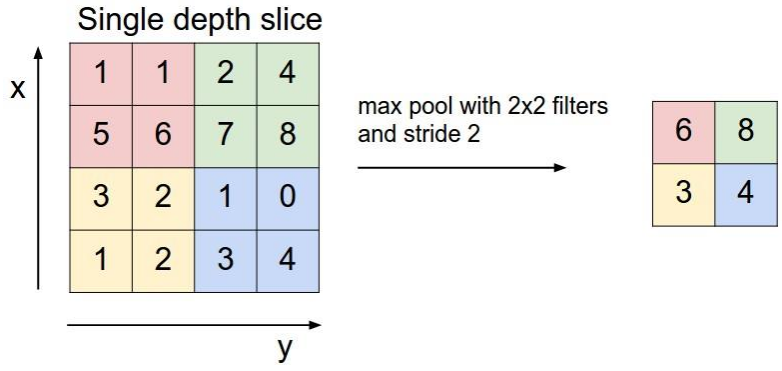
U konvolučních sítí lze dále použít jako aktivační funkci hyperbolickou tangentu, nebo sigmoitu, experimentálně bylo však zjištěno, že při použití ReLU je trénování sítě několikanásobně rychlejší [9].

4.1.3 Pooling vrstva

Pooling vrstva patří mezi vrstvy, které nepoužívají žádné neurony čili slouží primárně k úpravě dat, které jsou mezi sebou předávány dvěma jinými vrstvami. Používá se na podvzorkování vstupu, to slouží ke snížení velikosti map příznaků.

[2] Zdroj: https://blog.acolyer.org/delving-deep-relu-jpeg/

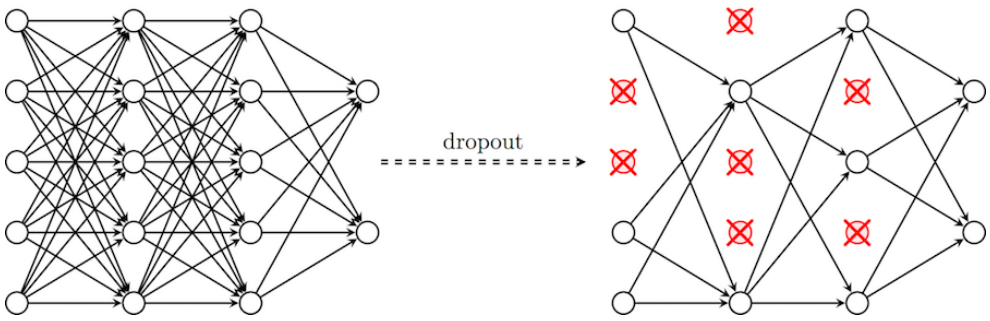
Max pooling aplikuje filtr na vstup a jako výstup udá největší číslo v oblasti, spadající do čtvercového pole filtru (jak je znázorněné na obrázku 7) o určené velikosti, nejčastěji 2x2, a kroku, který říká o kolik bodů se musí pole posunout pro získání dalšího bodu výsledku, též nejčastěji 2 [10]. Slouží k tomu, aby se předešlo jevu přetrénování sítě, kdy je síť velmi dobře naučená na trénovacích snímcích, ale nedokáže dobře klasifikovat testovací snímky, tedy ztrácí schopnost generalizace. Dalším užitím je snížení časové a paměťové náročnosti sítě pomocí snížení rozlišení vstupů.



Obrázek 7: Fungování max pooling vrstvy. Zdroj: [10]

4.1.4 Dropout vrstva

Dropout vrstva slouží k snížení přetrénování konvoluční sítě a lepší generalizaci příznaků. Použitím této vrstvy se v průběhu každé iterace učení, pomocí parametru pravděpodobnosti, náhodně deaktivují některé neurony [11]. Tím se dosáhne toho, že síť zapomene část vlastností, kterou se dříve naučila. Princip je znázorněn na obrázku 8.



Obrázek 8: Fungování Dropout vrstvy[3].

[3] Zdroj: https://sefiks.com/2018/03/19/handling-overfitting-with-dropout-in-neural-networks/

4.1.5 Plně propojená vrstva

Plně propojená vrstva (Fully Connected layer) slouží k tomu, aby využila toho, co se síť naučila, na určení pravděpodobnosti zařazení snímku do jednotlivé třídy [11]. Již podle názvu je zřetelné, že všechny neurony této vrstvy jsou propojené s každým neuronem z vrstvy předcházející. Při klasifikačních úlohách se kombinují naučené vlastnosti, za účelem určení kategorie snímku. To je důvod, proč má poslední plně propojená vrstva počet kanálů roven počtu kategorií.

4.2. Architektury konvolučních neuronových sítí

V této sekci jsou popsané některé úspěšné architektury neuronových sítí.

LeNet

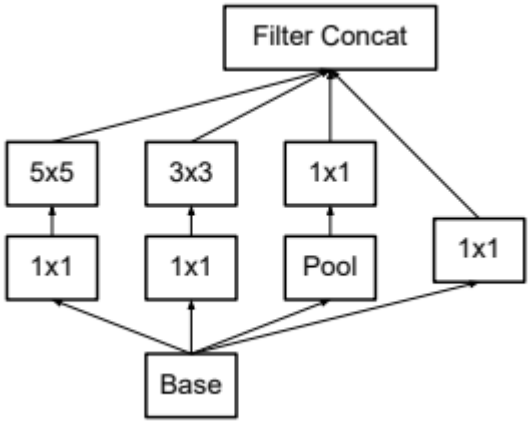
První úspěšná implementace konvoluční neuronové sítě byla vytvořena Yann LeCunem [12]. Byla primárně navržena pro rozpoznávání znaků v dokumentech. Poslední verze, *LeNet-5*, je tvořena osmi vrstvami – vstupní vrstvou, následovanou dvěmi konvolučními vrstvami, doplněnými o podvzorkovávací vrstvy a nakonec dvě plně propojené vrstvy ústící do výstupní vrstvy.

AlexNet

Tato síť byla představena v roce 2012 Alexem Krizhevskym se kterou vyhrál soutěž ILSVRC, což bylo zásádním průlomem pro výzkum konvolučních neuronových sítí. Oproti *LeNet* měla tato architektura, více vrstev a neuronů. Dále je speciální tím, že měla po sobě jdoucích několik konvolučních vrstev, bez následování podvzorkovací vrstvou, což bylo do té doby neobvyklé [13].

GoogleNet

V roce 2014 vyhrál ILSVRC Szegedy et al. s architekturou *GoogLeNet*. Zásadní přínos této architektury byl v tzv. *Iception Module* (obrázek 9), pomocí kterého se rapidně snížil počet parametrů sítě, *GoogLeNet* měla jen čtyři miliony parametrů oproti *AlexNet* se šedesáti miliony [14]. Jedná se o paralelní kombinaci 1x1, 3x3, 5x5 konvolučních filtrů, s předzpracováním pomocí 1x1 konvoluce pro redukci množství příznaků. Výhodou je to, že se nemusíme mezi použitými rozměry filtrů v jednotlivých vrstvách rozhodovat, ale necháme rozhodnout síť, které příznaky využije [15].



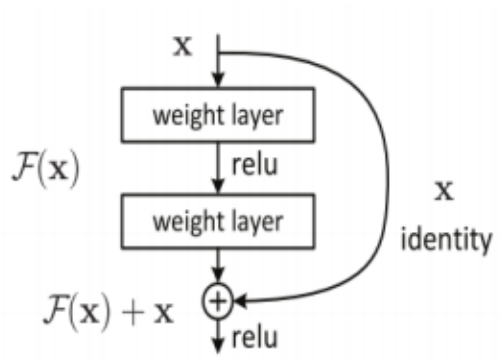
Obrázek 9: Schéma Iception modulu. Zdroj: [14]

VGG

Tato síť byla vyvinuta pomocí Karena Simonyana a Andrewa Zissemana na Oxfordské univerzitě a na ILSVRC obsadila v roce 2014 druhé místo. Ti ve svém článku z roku 2015 představili sérii experimentů na kterých demostrují, že hloubka sítě má zásadní vliv na rychlost [16]. VGG měla vrstev celkem šestnáct. Dále využívala v konvolučních vrstvách filtry o velikosti kernelu 3x3, tedy o mnoho menší nežli *AlexNet*.

ResNet

Tato síť vyhrála ILSVRC v roce 2015. S narůstající hloubkou sítě, pro přesnější klasifikaci, vzinkl problém zhoršení trénovací a testovací chyby. Za pomocí experimentů se zjistilo, že sítě s větším počtem vrstem dosahují vyšší chybu nežli sítě s niším počtem vrstev [17]. Autoři článku z roku 2015 [17] navrhli reziduální blok (obrázek 10), který je základním stavebním blokem této sítě.

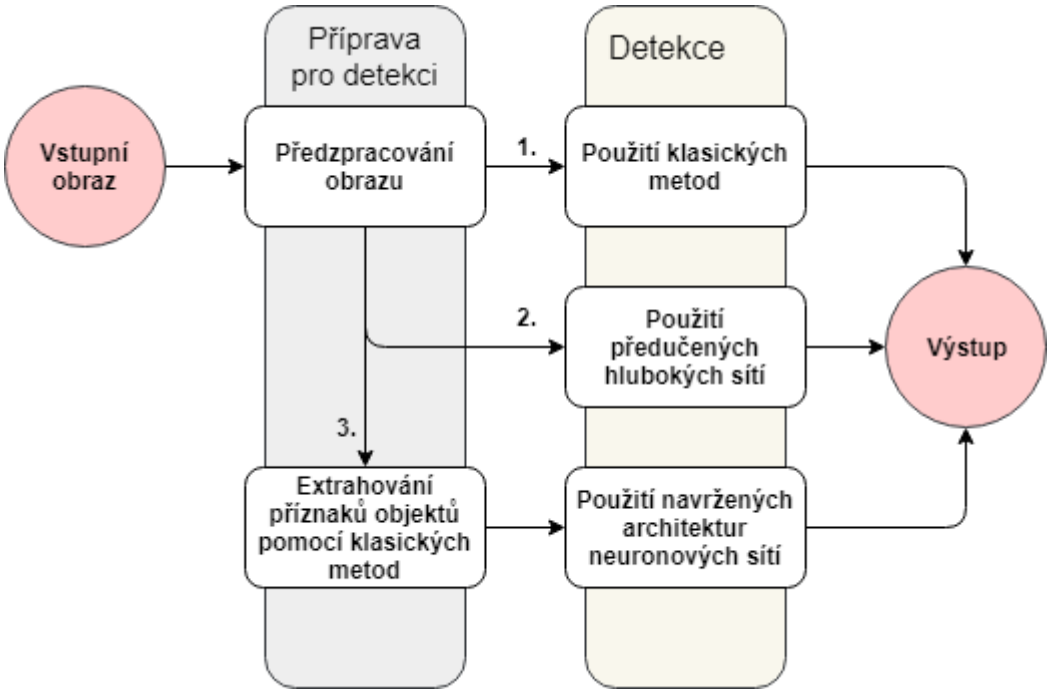


Obrázek 10: Reziduální blok. Zdroj: [17]

Budeme-li mít x jako vstup do podsítě a její výstup bude $H(x)$. Zbytkovou funkcí tím pádem bude rozdíl mezi nimi $F(x) = H(x) – x$, původní funkce je tedy vyjádřitelná tvarem $H(x) = F(x) + x$. Přestože můžeme oba zápisy zaměnit, složitost optimalizace je při trénování zbytkové funkce odlišná [17].

5 Návrh řešení

Při návrhu řešení se postupovalo ve třech rovinách, to z oblasti strojového vidění i neuronových sítí (obrázek 11).



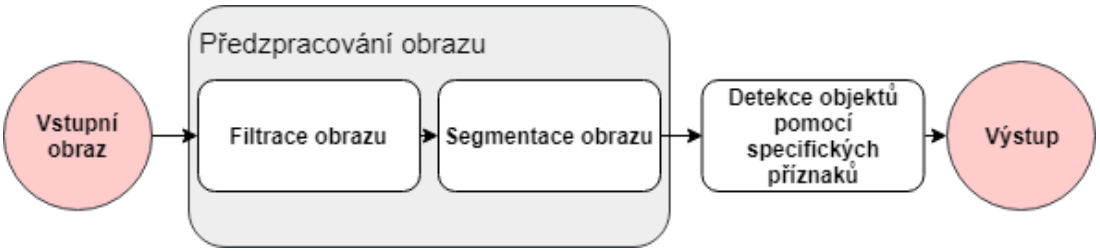
Obrázek 11: Schéma přístupu k řešení problému detekce.

Pro veškeré programování bylo vytvořeno virtuální prostředí v programu **Annaconda3,** pod operačním systémem **Windows 64-bit**, s použitím programovacího jazyka **Python** ve verzi **3.6**, která byla zvolena díky kompatibilitě s frameworkem Tensorflow (ku příkladu Python 3.7 prozatím podporován není) a vysoké dostupnosti knihoven pro analýzu dat a zpracování obrazu. Pro práci s klasickými metodami byla užita knihovna **OpenCV** (popsána v kapitole 3.2) ve verzi **4.0.0.21**.

5.1 Klasické metody

U klasických metod bylo k návrhu přistupováno prvotním zkoumáním metod předzpracování obrazu, za účelem aplikovat je tak, aby bylo možné pomocí vhodných metod detekovat roztoče na bázi jejich specifických příznaků (obrázek 12).

Snímky, které byly pořízeny, jak je popsáno v kapitole 3.1, byly pro další zpracování v oblasti strojového vidění ponechány v nezměněné formě a náhodně vybrány pro účely testování úspěšnosti klasických metod.



Obrázek 12: Schéma postupu detekce objektu za užití klasických metod.

5.1.1 Tvar

Prvotní myšlenkou bylo roztoče detekovat pomocí jeho specifického elipsového tvaru. Zde byl jako filtrace užit převod snímku na stupně šedi. Knihovna OpenCV umožňuje detekci objektů na základě tvaru pomocí funkce **SimpleBlobDetector**, která funguje na principu:

a) Převod zdrojových snímků na několik binárních pomocí zvoleného prahu.

b) V každém binárním snímku jsou pro bílé pixely detekovány souvislé oblasti.

c) Jsou vypočítány středy shluků souvislých oblastí a oblasti blíže ke středu jsou spojeny .

d) Středy a úhel nové oblasti jsou vypočítány a navráceny jako nová hodnota.

Celý tento algoritmus je kontrolován pomocí parametrů uvedených v tabulce 1, u všech parametrů je možno nastavit maximální a minimální hodnoty. K ilustaci hodnot použitých v této práci slouží obrázek 13.

Parametry	Popis parametrů
Treshold	hodnota prahu
filterByArea	velikost kolekce v obrazových bodech
filterByCircularity	hodnota přiblížení kruhu (0-1)
filterByConvexity	úplnost objektu (0-1)
filterByInertia	smrsknutí objektu (0-1)

Tabulka 1: Parametry funkde SimpleBlobDetector.

```
# Setup SimpleBlobDetector parameters.
params = cv2.SimpleBlobDetector_Params()

#Change thresholds
params.minThreshold = 1
params.maxThreshold = 121

# Filter by Area.
params.filterByArea = True
params.maxArea = 120

# Filter by Convexity
params.filterByConvexity = True
params.minConvexity = 0.7

# Filter by Inertia
params.filterByInertia = True
params.maxInertiaRatio = 0.7
```

Obrázek 13: Experimentálně doporučené nastavení parametrů funkce.

5.1.2 Barva

Druhou myšlenkou bylo roztoče od pozadí snímku oddělit pomocí jeho specifické barvy. K tomu byla využita filtrace pomocí změny modelu BGR na HSV, který je popsán v kapitole 3.2.1. Na základě této filtrace byla provedena segmentace obrazu pomocí spodní a horní hodnoty vybrané barvy v tomto barevném modelu a vytvořena ořezová maska. V té bylo následně experimentováno s redukcí šumu pomocí Gaussového vyhlazení a mediánovým filtrem, kde se výhodnějším filtrem pro redukci okolního / náhodného šumu, po experimentování, jevil mediánový filtr a tak jej bylo využito pro další testování a vyhodnocení úspěšnosti. Následná detekce je provedena pomocí funkce knihovny OpenCV **findContours()**, která ukládá do pole nalezené body obrysu a má tři parametry:

a) vstupní obrázek na kterém bude detekce obrysů provedena

b) hiearchie obrysů

c) princip zachování bodů obrysu

Podrobný popis parametrů a jejich hodnoty jsou uvedeny v experimentální části práce.

5.2 Neuronové sítě

Při návrhu řešení pomocí neuronových sítí byly vzaty v potaz, jak předučené sítě, tak i navržení architektur konvolučních sítí.

Za účelem získání dat pro datasety neuronových sítí se postupovalo dvěma směry. Pro vytvoření pokusů za pomocí předučené neuronové sítě byli roztoči na snímcích označeni manuálně za použití programu LabelImg.

Pro navržené architektury byli roztoči vyříznuti ze snímků a spolu s nimi i jiné náhodné objekty, jako včely, měl, špína na podložce, více v kapitole 5.2.2, aby vznikly dvě klasifikační třídy (objekt, který je roztoč, a objekt, který roztoč není).

5.2.1 Použité vybavení

Učení všech testovaných modelů probíhalo na autorově osobním počítači. Součástí sestavy je procesor Intel i5-7300HQ, grafická krata NVIDIA GeForce GTX 1050 Ti a 8 GB paměti RAM. Bližší specifikace uvedených komponent v tabulkách 2 a 3.

Počet jader / vláken	4 / 4
Základní frekvence / Boost frekvence	2.5GHz / 3GHz

Tabulka 2: Procesor.

Výpočetní jednotky	32
Maximální výkon	2,14 TFLOPS
Paměť	4GB GDDR5
Max. šířka paměti	112 GB/s

Tabulka 3: Grafická karta.

S ohledem na možnost využití modelů pro mobilní aplikaci bylo experimentování a následné testování navržených architektur prováděno pouze za použití slabšího procesoru a 4 GB paměti RAM. Blížší specifikace procesoru je uvedena v tabulce 4.

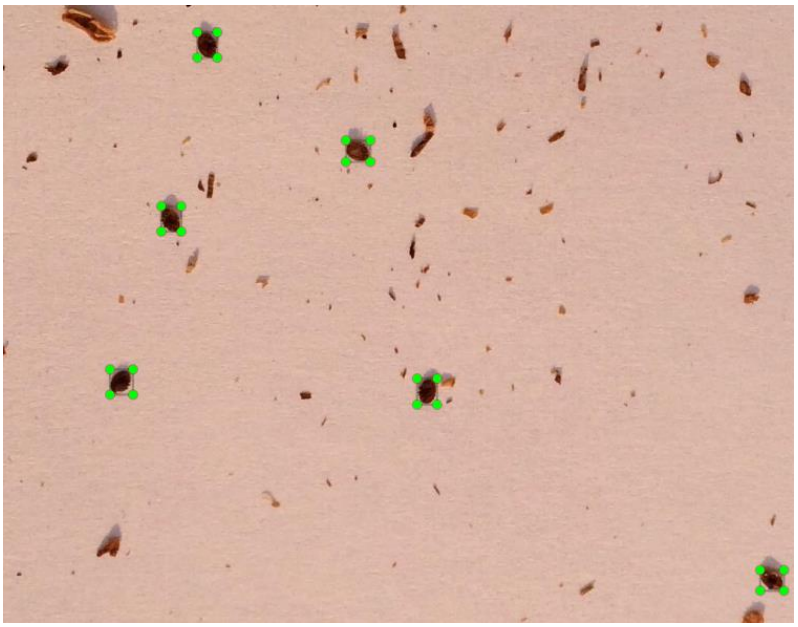
Počet jader / vláken	2 / 4
Maximální frekvence	2.5GHz

Tabulka 4: Procesor pro testování neuronových sítí.

5.2.2 Předučené sítě

V duchu použití osvědčené architektury bylo k řešení přistupováno pomocí transfer learningu (předučené sítě). Pro zachování kvality nebylo rozlišení pořízených snímků redukováno. Místo toho byly snímky nařezány na devět stejných částí, použitím knihovny imageSlicer, o přijatelném rozlišení pro vstup neuronové sítě. Manuálním označením nařezaných snímků vzniknul dataset pro trénování detekce roztoče,

popsaný tabulkou 2. Vzory v tomto datasetu jsou souřadnicemi obdélníkového rámečku, ohraničujícího roztoče v obraze (obrázek 14).



Obrázek 14: Manuálně vyznačení roztoči pomocí programu LabelImg.

Označená data byla prvotně promíchána za pomocí modulu random, s funkcí seed() nastavenou na hodnotu 42. K rozdělení dat do dvou množin došlo za použití funkce train_test_split() z knihovny scikit-learn, 75% vzorů bylo přiřazeno do trénovací a 25% do testovací množiny. Ukázka použité funkce je přiložena v obrázku 15.

```
(trainX, testX, trainY, testY) = train_test_split(data,
    labels, test_size=0.25, random_state=42)
```

Obrázek 15: Ukázka užité funkce pro rozdělení dat.

Množina	Počet vzorů
Trénovací	2499
Testovací	883

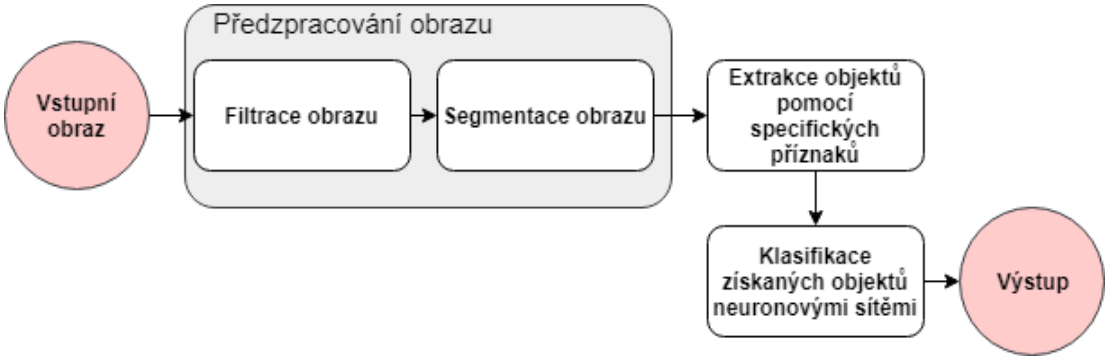
Tabulka 5: Dataset pro předučené sítě.

Pro tento návrh bylo použito TensorFlow Object Detection API, ke trénování předtrénovaného modelu architektury InceptionV4, celý proces byl inspirován již existujícím projektem [18]. Podrobný seznam knihoven pro funkci programu je popsán na stránce projektu. Jako backend byl v této práci nainstalován **Tensorflow-gpu** ve

verzi **1.12.0**. Ten byl vybrán z důvodů kompability s grafickou kartou NVIDIA. Pro využití plného potencionálu grafické karty byla nainstalována knihovna pro hluboké sítě **cuDNN** ve verzi **7** a software **CUDA Toolkit** ve verzi **9**.

5.2.3 Navržené architektury neuronových sítí

Za účelem nižší náročnosti a výpočetního času programu bylo k řešení přistupováno navržením architektur, které slouží jako rozšíření klasických metod a obstarávají klasifikaci detekovaných objektů (obrázek 16).

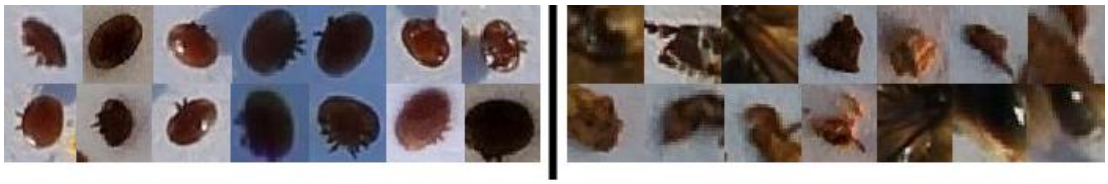


Obrázek 16: Schéma detekce objektu za použití navržených architektur neuronových sítí.

Předzpracování obrazu a extrakci objektů pro vstup neuronové sítě obstarává řešení popsané v kapitole 5.1.2. Při segmentaci obrazu byla spodní a horní hodnota barvy barevného modelu nastavena tak, aby byly do dalších kroků algoritmu získány i všechny objekty, které není možné získat bez výskytu False Positive objektů. To jsou objekty, které nemají být klasifikovány jako správné, ale v našem případě je extrakce jako správné považuje. Tento výstup je dále použit jako vstup neuronové síti a ta určí správnost extrahovaných objektů. Správně klasifikované objekty spočítá a vykreslí jejich lokalizaci na vstupní obraz.

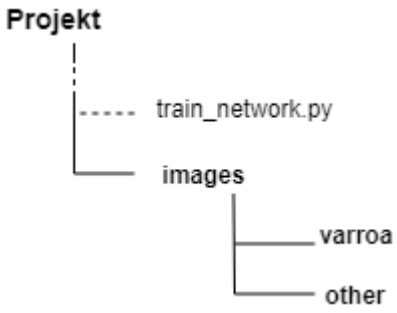
5.2.2.1 Dataset

Rozmanitostí vstupních dat neuronové sítě bylo dále inspirováno při vytvoření datasetu popsaného tabulkou 6. Byly vytvořeny dvě klasifikační třídy *varroa* a *other*, které reprezentují získaná data z předešlých kroků algoritmu. Třída varroa obsahuje snímky roztoče vystřižené z pořízených snímků a třída other je naplněna snímky na kterých jsou měl, části včel a další objekty, které by extrakcí mohly nastat jako vstup neuronové sítě. Pro znázornění datasetu slouží obrázek 17.



Obrázek 17: Ukázka datasetu. Levá část je třída *varroa* a prává *other*.

Označení snímků je zde řešeno pomocí fyzického uložení snímků, kdy si skript pro učení neuronové sítě, který je umístěn ve stejném adresáři jako adresáře s vystřiženými snímky, vezme název adresáře a jedná-li se o *varrou* přiřadí datům z adresáře 1, v případě *other* je přiřazena 0. Struktura adresářů a ukázka kódu, který je součástí skriptu pro učení neuronové sítě, jsou demonstrovány na obrázku 18 a 19.



Obrázek 18: Struktura adresářů projektu.

```
labels = []
for imagePath in imagePaths:
    #extract label from the path and update the labels list
    label = imagePath.split(os.path.sep)[-2]
    label = 1 if label == "varroa" else 0
    labels.append(label)
labels = np.array(labels)
```

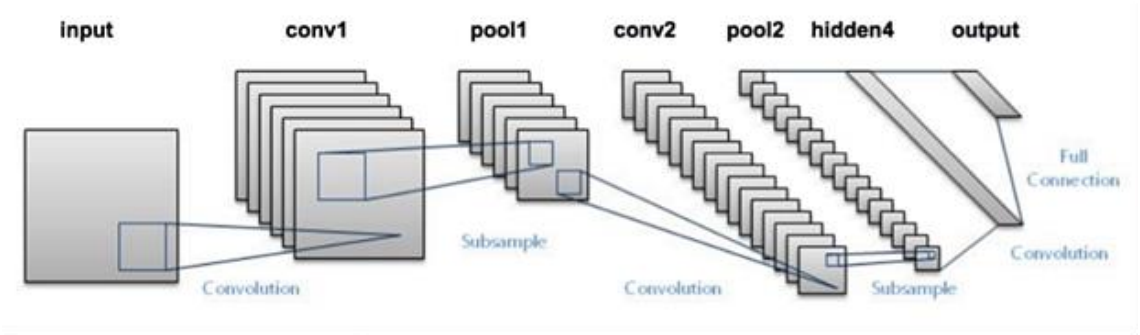
Obrázek 19: Ukázka kódu pro označení snímků.

Název třídy	Množina	Počet vzorů
Varroa	Trénovací	743
	Testovací	248
Other	Trénovací	511
	Testovací	171

Tabulka 6: Dataset pro učení a testování neuronových sítí.

5.2.2.2 Skripty

Jako inspirace pro návrh architektur neuronových sítí byla, kvůli jednoduchosti a nenáročnosti na výkon, architektura *LeNet-5* [12], její schéma je uvedeno na obrázku 20.



Obrázek 20: Schéma architektury LeNet-5[4].

Byly navrženy celkem tři sítě, které jsou blíže popsány v experimentální části práce. Každá síť má vytvořený vlastní projekt, jehož součástí jsou tři skripty:

`network.py` – Zde je vytvořena třída, se statickou metodou build() se čtyřmi parametry:

- šířka vstupního obrazu
- výška vstupního obrazu
- barevná hloubka –> počet kanálů ve vstupním obraze (např. 3 pro RGB)
- počet klasifikačních tříd

V této metodě je dále vytvořený sekvenčí model, do kterého jsou přidány jednotilivé vrstvy sítě.

`train_network.py` – Tento skript slouží pro samotné trénování sítě. Má definované tři volitelné argumenty:

- **dataset** – Cesta k datasetu použitému pro trénování sítě.
  - Výchozí hodnota je nastavena podle obrázku 18.
- **model** – Jméno a cesta, kam bude uložen natrénovaný model.
  - Výchozí hodnota je nastavena *varroa.model* a uložení je ve stejném adresáři jako train_network.py
- **výstupní graf** – Jméno a cesta, kam chceme uložit graf průběhu učení sítě.
  - Jako výchozí hodnota je nastaven graf.png ve stejném adresáři.

[4] Zdroj: https://blog.dataiku.com/deep-learning-with-dss

Dále je do něj načtena třída z `network.py` a z ní je natrénován model s příslušným optimalizátorem a doplněním parametrů třídy.

Skript se také stará o označení snímků a rozšíření datasetu, použitím funkce ImageDataGenerator() importované z *keras.preprocessing.image*. Ta se užitím různých transformací vstupních snímků stará o úspěšnější trénování v souvislosti s menším objemem dat v datasetu.

`test_network.py` – Slouží k samotnému provedení detekce. Má jeden volitelný argument, kterým je cesta k natrénovanému modelu a název modelu. Jeho výchozí hodnota je, pro předchozí uložení modelu ve stejném adresáři, pouze *varroa.model*.

Obsahuje algoritmy pro načtení snímku, extrahování objektů, spadajících do daného intervalu hodnot barevného modelu HSV, které poté klasifikuje užitým natrénovaným modelem, spočítá a vykreslí na vstupní snímek.

5.2.2.3 Použité knihovny

Pro vývoj neuronových sítí bylo využito knihovny Keras s frameworkem Tensorflow-gpu, pro testování byl použit framework Tensorflow, využívající pouze procesor. K vizualizaci dat pro výstupní graf byla užita knihovna Matplotlib. Podrobný seznam použitých knihoven v této práci je popsaný v tabulce 7.

Knihovna	Verze
Keras	2.2.4
Tensorflow-gpu	1.12.0
Tensorflow	1.12.0
Numpy	1.16.0
Pandas	0.23.4
Pillow	5.4.1
Matplotlib	3.0.2
Scikit-learn	0.20.3

Tabulka 7: Knihovny a jejich verze využité v této práci.

6 Experimenty

V této části jsou popsány konkrétní experimenty a jim příslušné výsledky.

6.1 Klasické metody

Cílem experimentů bylo prozkoumat metody popsané v kapitole 5.1. Jejich výstupy sice nevedly k samostatně použitelným řešením, avšak dobře posloužily k pochopení celé problematiky detekce roztoče. Metoda detekce podle barvy byla, díky jejim výsledkům a možnosti generalizace, dále využita jako předzpracování obrazu a získání vstupu pro navržené architektury neuronových sítí.

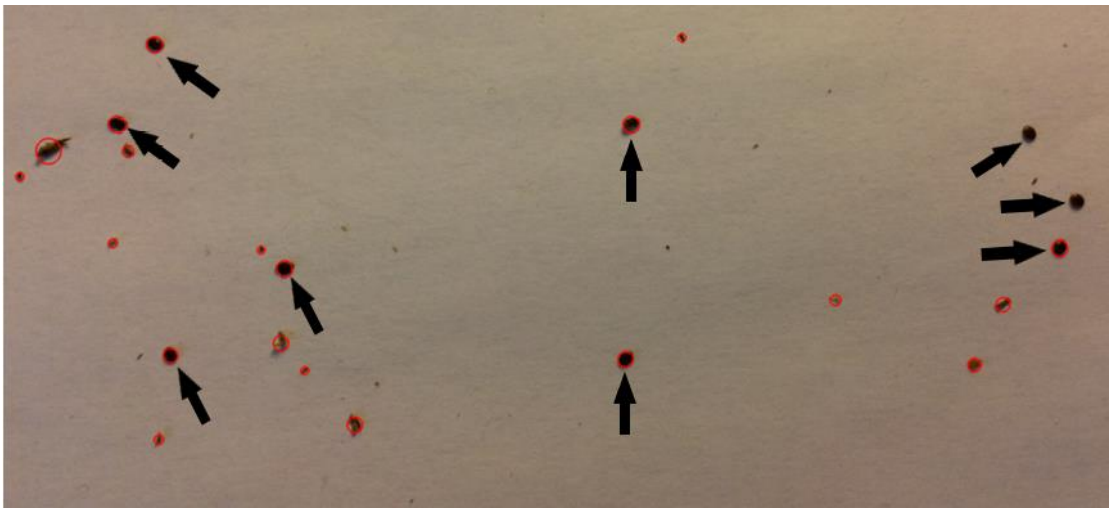
6.1.1 Tvar

V první etapě bylo experimentováno s funkcí SingleBlobDetection a primárním cílem bylo zjištění možnosti generalizace jejího použití pro široké spektrum vstupních snímků. Prvotně byl snímek převeden na stupně šedi. Takto připravený snímek byl poté použit jako vstup pro funkci a dále již bylo experimentováno pouze s nastavením parametrů funkce.

Experimentováním s nastavením parametrů (popsaných v tabulce 1) bylo získáno vcelku uspokojivé řešení pro snímky, které zahrnovaly vyfocenou celou podložku a byly vyfoceny v přibližně stejné vzdálenosti od podložky. Správně nalezených roztočů bylo 94%. Nastavení těchto parametrů je uvedeno na obrázku 13. I přes to se stále vyzkytuje poměrně velká množina objektů, které jsou detekovány jako roztoči, i když jimi nejsou. Při širším aplikování funkce s takto nastavenými parametry, však detekce ztrácí možnost generalizace a je značně nepřesná, čtenář si může udělat představu pomocí výsledků na obrázku 21 a obrázku 22.

Nutno dodat, že pro užití funkce na různě pořízených snímcích bylo upuštěno od parametru, udávajícího maximální velikost objektu. Obrázek 21 je výstupem funkce, která má v sobě tento parametr zahrnut, a i v tomto případě nejsou detekovány všechny správné objekty, při jeho vypuštění byly také detekovány i další části včel.

Obrázek 11 : Detekce na snímku z celé podložky s ideálně nastavenými parametry. Objekty spadající do množiny detokovaných roztočů jsou označeny červeným kruhem.



Obrázek 22: Detekce na přiblíženém snímku. Objekty spadající do množiny detekovaných roztočů jsou označeny červeným kruhem. Šipkami jsou označeni roztoči, kteří měli být detekováni.

S různorodostí roztočů, jako je například otočení roztoče na bok a deformace tvaru roztoče, aplikovaná funkce na výstupu udává více False positive výsledků a do množiny detekovaných roztočů nejsou zahrnuty objekty, které jsou v ní očekávány.

Z těchto důvodů bylo od této metody detekce upuštěno a dále nebyla užita ani jako předzpracování obrazu pro neuronovou síť. Avšak toto řešení by mohlo najít uplatnění u pořízení snímku, s předem udanou vzdáleností pořizovacího zařízení od snímané podložky.

6.1.2 Barva

V druhé etapě bylo experimentováno s metodou detekce založenou na barvě roztoče. Pro tento experiment byl vstupní snímek převeden do barevného modelu HSV. Podle jeho barvy bylo nastaveno rozmezí, ve kterém se daná barva vyskytuje (obrázek 23).

```
lowerH = np.array([0,50,12])
upperH = np.array([11,255,255])
```

Obrázek 23: Mezní hodnoty barvy roztoče (v pořadí H,S,V).

Pomocí těchto dvou hodnot byla vytvořena ořezová maska a na ni byl uplatňen mediánový filtr s kernelem o rozměrech 3x3 pro odstranění náhodného šumu. Tato maska byla dále použita jako vstup pro funkci findContours s nastavenými parametry, které jsou ilustrovány na obrázku 24.

```
cv2.findContours(median.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Obrázek 24: Ukázka užité funkce.

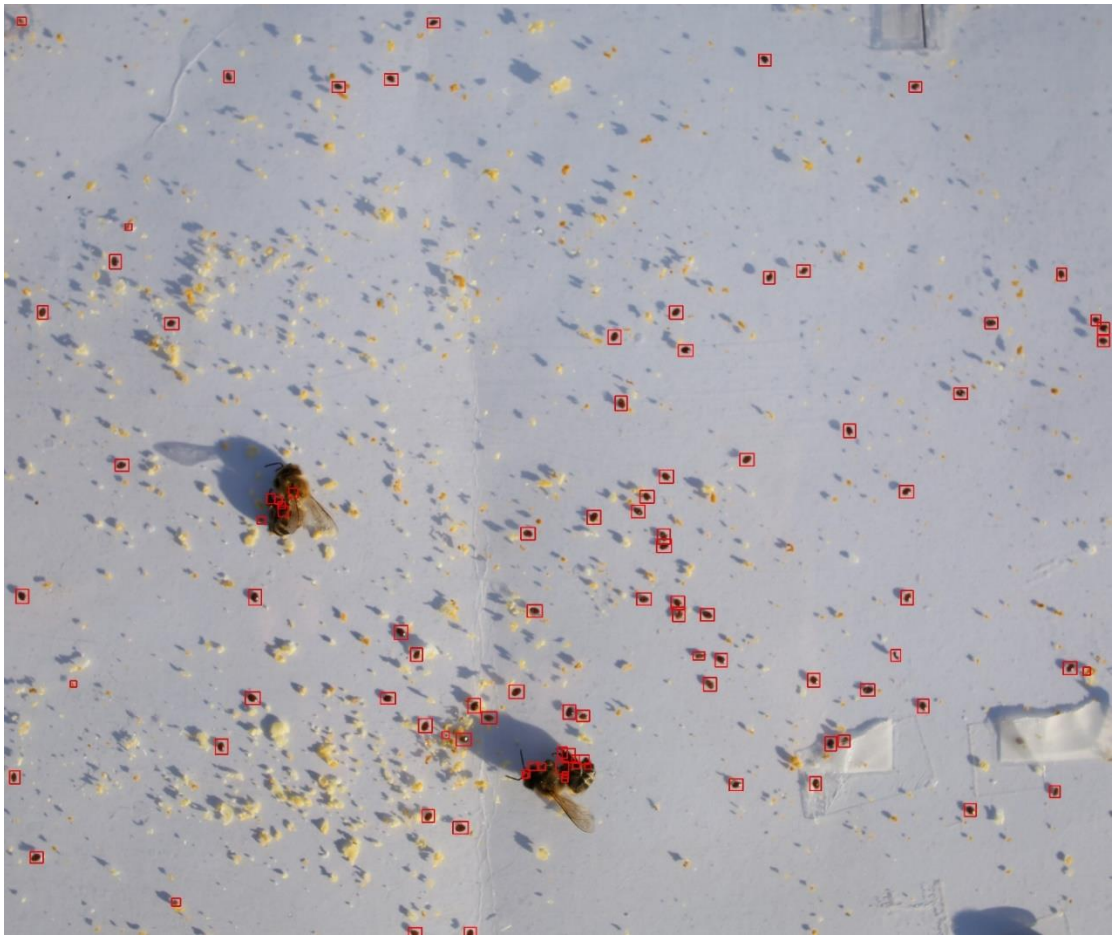
Kde prvním nastaveným parametrem je ořezová maska. Druhý parametr udává, že se nebude jednat o detekci vnořených objektů, to značí, že u případné detekce potomka se bude brát na zřetel pouze rodič. Třetí parametr je nastaven na to, aby za účelem šetření paměti a z důvodů nutnosti pouze dvou klíčových bodů, to souřadnice začátku obrysu a konce obrysu, uchoval pouze tyto dvě hodnoty. Pole nalezených obrysů je následně v cyklu procházeno a jsou nad ním na vstupní snímek vykresleny, za užití funkce **rectangle(),** křivky, které ohraničují nalezené objekty.

```
1  #loop over found contours
2  for i in range(len(contours)):
3      #load coordinates of contours
4      x,y,w,h=cv2.boundingRect(contours[i])
5      #write rectangle around detected object
6      cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255), 1)
```

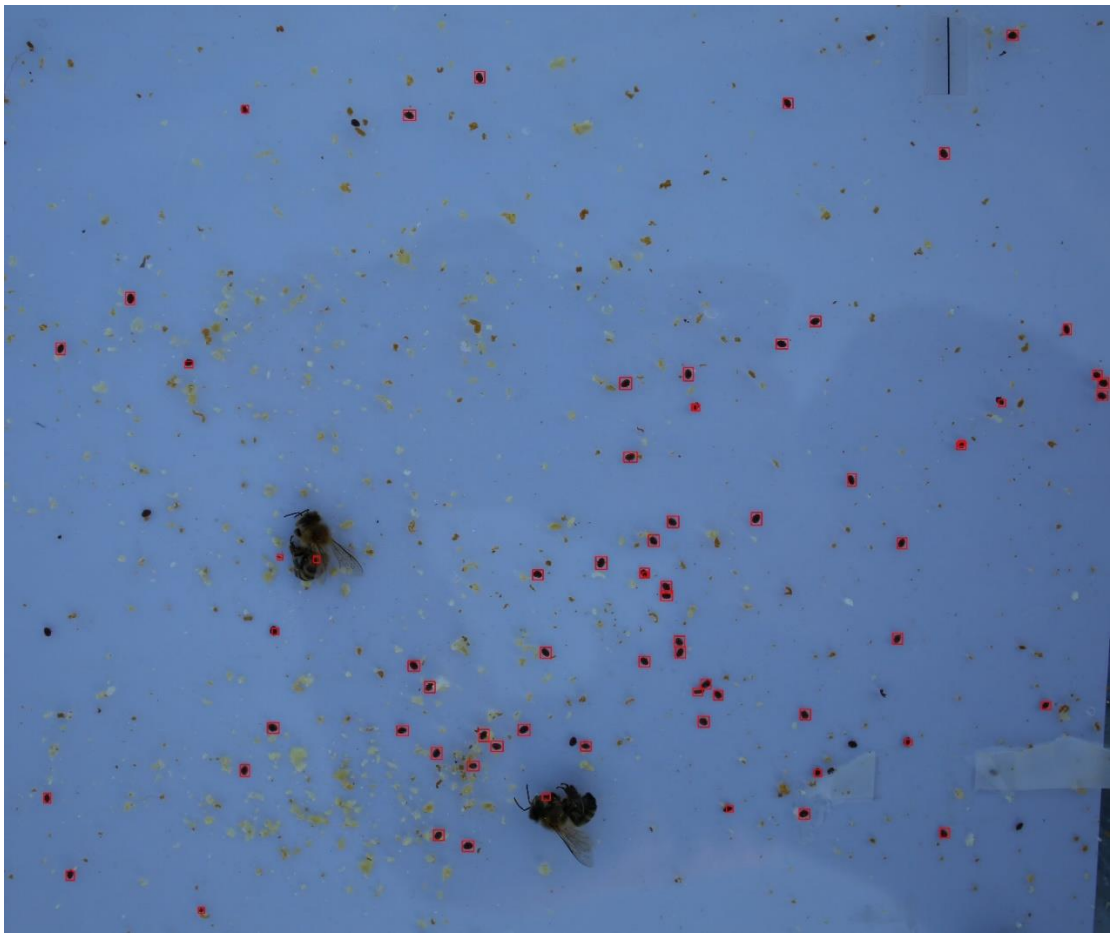
Obrázek 25: Ukázka vykreslení ohraničení objektů.

Výstupy experimentu vykazují značně uspokojivé výsledky, spíše však pro další zpracování neuronovými sítěmi, jelikož se stále vyskytuje i značný počet False positive objektů. K porovnání s předchozí metodou slouží obrázek 26.

Generalizace je, díky segmentaci a detekci založené na barvě roztoče, zachována a ve vstupních snímcích jsou detekovány všechny objetky, které by měly spadat do množiny správných objektů. Výjimkou jsou snímky, které byly pořízeny za horších světelných podmínek, například při šeru (obrázek 27), kdy část roztočů není z neznámého důvodu do intervalu barvy zařazena. Avšak počet těchto nedetekovaných roztočů spadá do tolerované hranice (tj. rozdíl sedmi roztočů) a tak bylo od dalšího zkoumání tohoto problému upuštěno ve prospěch správné klasifikace nalezených objektů pomocí neuronových sítí.



Obrázek 26: Detekce založená na barvě roztoče. Objekty spadající do množiny detekovaných roztočů jsou vyznačeny čtverci.



Obrázek 27: Ukázka detekce na snímku s horšími světelnými podmínkami. Objekty spadající do množiny detekovaných roztočů jsou vyznačeny čtverci (tmavé elipsové objekty, které nejsou označeny, jsou roztoči, kteří do této množiny nebyli z důvodů horších světelných podmínek zahrnuti).

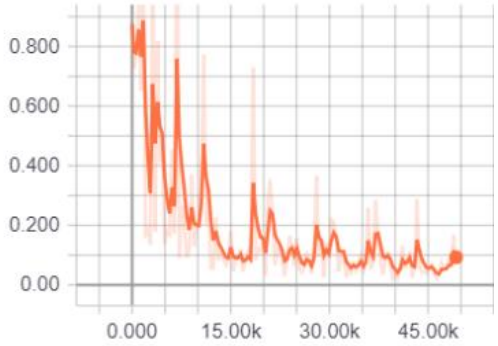
6.2 Neuronové sítě

Zde jsou popsány experimenty nad předučenou sítí a navrženými architekturami spolu s jejich vyhodnocením.

6.2.1 Předučená síť

Experimenty z oblasti předučených sítí byly provedeny za užití předtrénovaného modelu InceptionV4, ten byl dotrénován na datasetu popsaném v kapitole 5.1.1. Model byl trénován, podle doporučení z projektu [18], dokud se chyba sítě nedržela stabilně pod hodnotou 0,1. To odpovídalo zhruba 50000 krokům. Průběh učení je reprezentován grafem na obrázku 28.

Síť byla validována za použití dvaceti snímků, které nebyly zahrnuty v trénovacím datasetu. Tyto snímky byly z důvodu zachování kvality, při zmenšení rozlišení síť ztrácela schopnost detekce, nařezány na devět stejných částí a nad každou z nich byla provedena detekce a vykreslení nalezených objektů. Výstupem programu byl poté opět poskládaný snímek. Síť dosahovala vcelku dostačující hodnoty, správně bylo detekováno 92 % roztočů a množina False positive zahrnovala malé množství objektů a to převážně včely, které tvarem jejich těla připomínaly roztoče. Celý tento proces však vykazuje značnou časovou náročnost, kdy s použitým hardwarem zahrnujícím grafickou kartu trvá i více než 20 sekund. Z těchto výsledků a z důvodů zbytečně velké hloubky sítě bylo od dalšího experimentování v oblasti předučených sítí upuštěno.



Obrázek 28: Průběh učení sítě IceptionV4.
Graf je výstupem vizualizační utility Tensorboard.

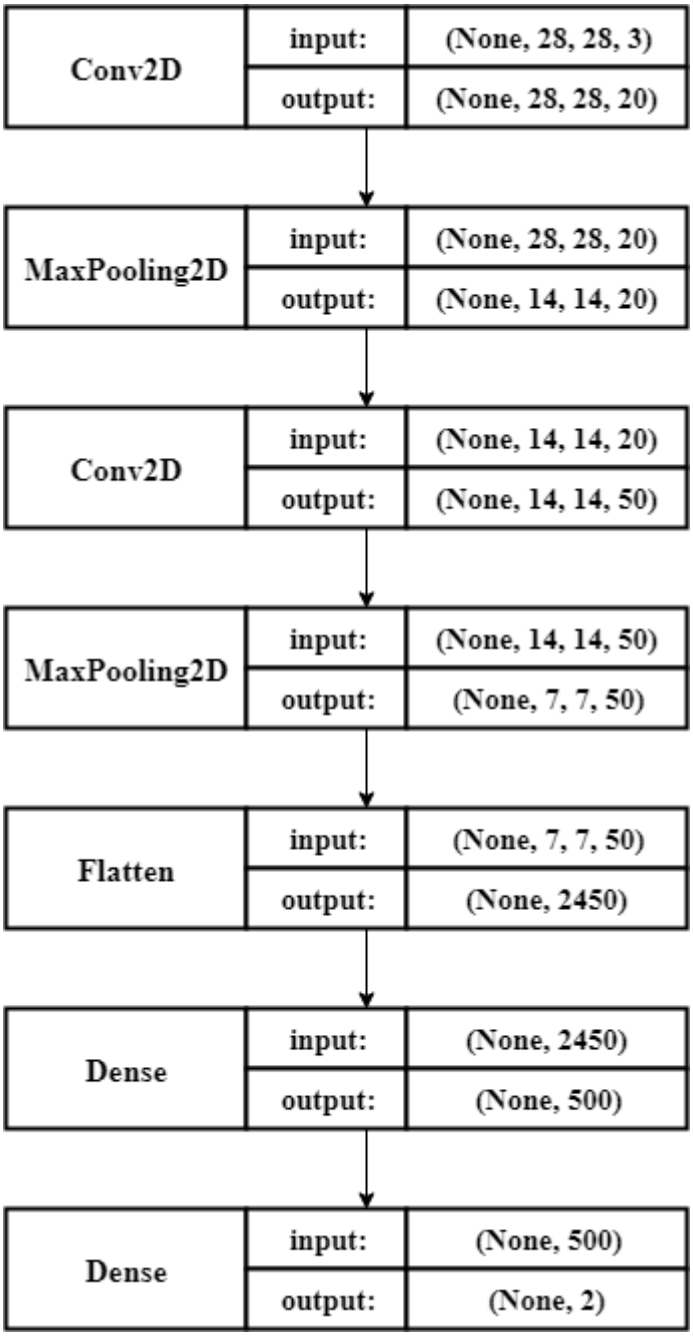
6.2.2 Navržené architektury neuronových sítí

Pro experimentováni v oblasti navržených architektur byly vytvořeny tři sítě. Hlavní stavební prvek těchto sítí byl opakující se blok konvoluční vrstvy s velikostí kernelu 5x5, aktivační funkce ReLU a maxpool vrstvy bez překryvu s velikostí kernelu 2x2 a posunem (z anglického *stride*) o dva obrazové body po ose x a ose y. Předposlední vrstvy byly plně propojené a zakončené vrstvou, která má dva výstupy, jimiž jsou dvě výsledné třídy. Konvoluční vrsty v sobě dále zavádí tzv. „same padding“, který zabraňuje redukování šířky a výšky jejich výstupu, pomocí orámování vstupního obrazu před filtrací, aby kernel mohl zpracovat i okrajové obrazové body.

Ve všech architekturách bylo dále experimentováno s dropout vrstvou s mírou výpadku 0,25 a 0,5. Konvoluční vrstvy byly také z vybraného počtu 20 filtrů v prvním a 50 v druhém bloku rozšířeny na 32 v prvním a 64 ve druhém. Obě tyto varianty

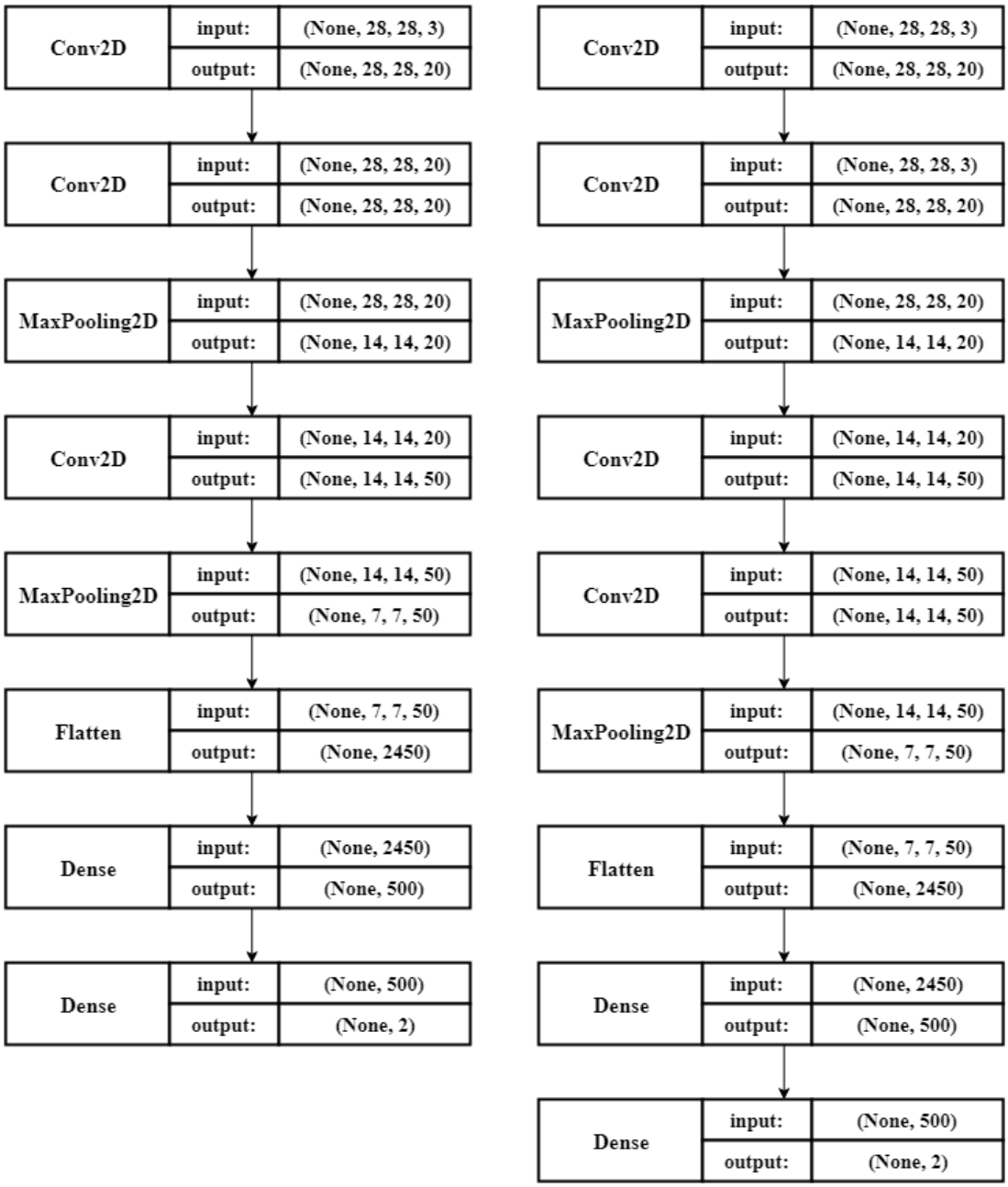
rozšíření sítě však vedly k tomu, že síť ztrácela svoje rozlišovací schopnosti a tak od nich bylo ve finálních architekturách upuštěno.

Jako první byla navržena síť popsaná schématem na obrázku 29. Účelem bylo ověřit, zda je možné s menším počtem vrstev v architektuře detekovat roztoče, či ne.



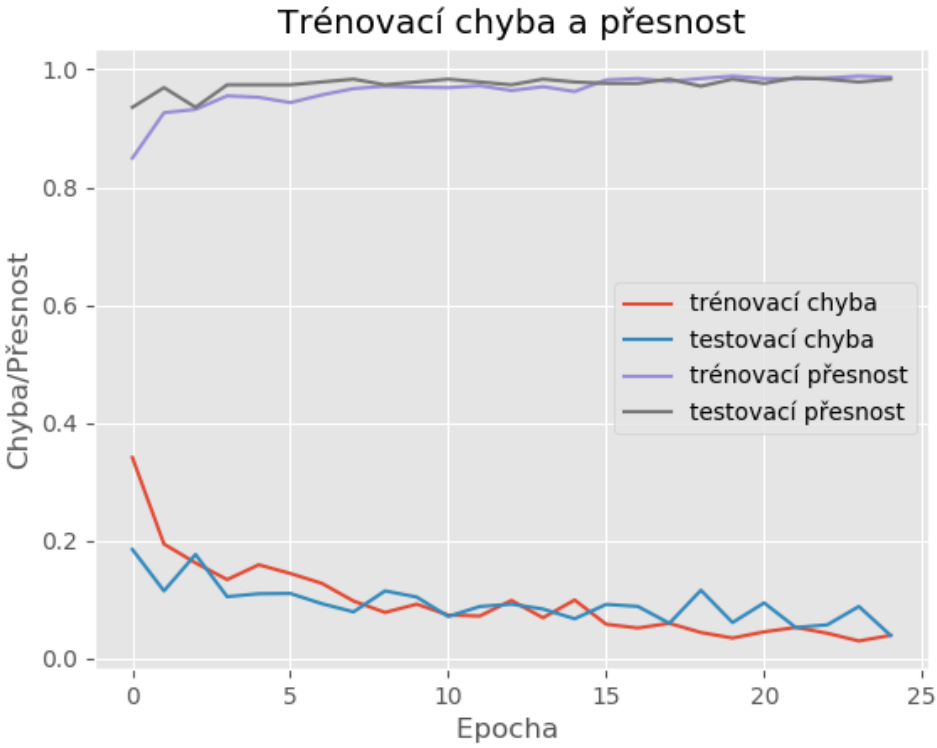
Obrázek 29: Schéma architektury první sítě.

Po ověření možnosti roztoče detekovat, s použitím malého počtu vrstev, bylo přistoupeno k experimentům v prohlubování sítě, kdy jsou do bloků přidávany konvoluční vrstvy. Schémata těchto sítí jsou popsány na obrázku 30.

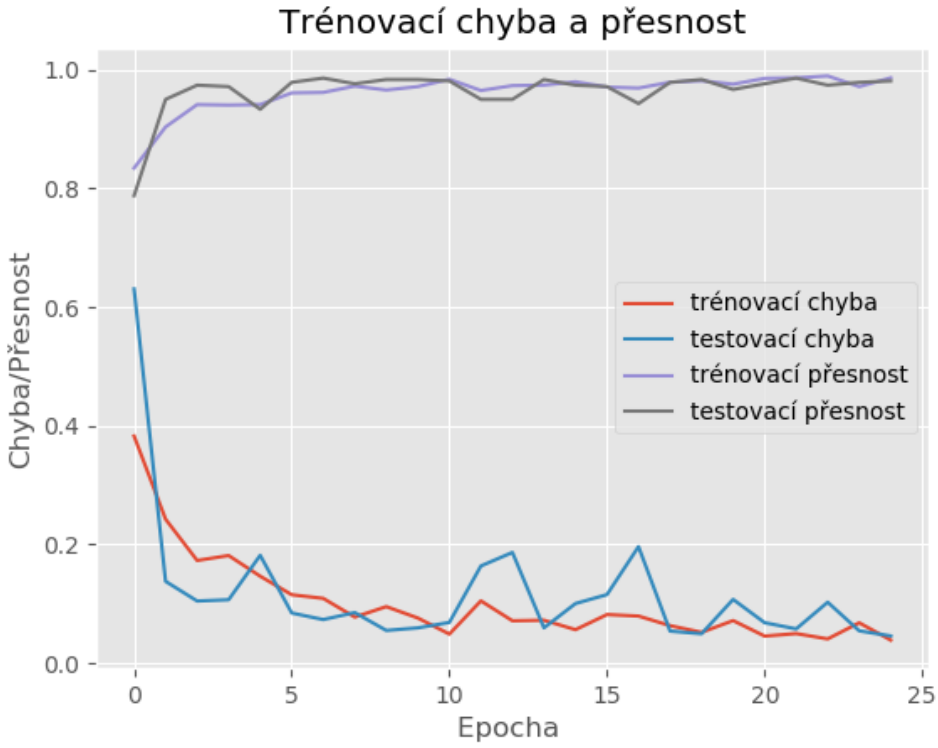


Obrázek 30: Schémata druhé a třetí sítě.

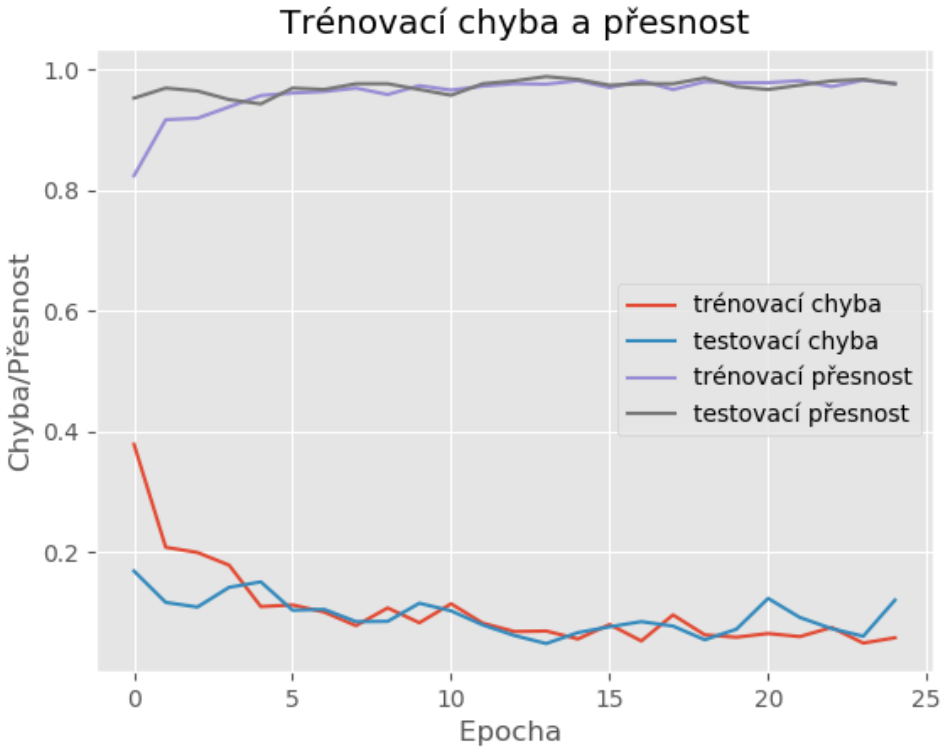
Navržené sítě byly učeny za použití datasetu popsaném v kapitole 5.2.2.1 na normalizovaných vstupech, optimalizačním algoritmem Adam, který adaptivně mění rychlost učení a rozšiřuje tak často používaný algoritmus klesání podle gradientu, s rychlostí učení 0,001 po 25 epoch. Jako chybová funkce byla použita binární cross-entropie (angl. *binary cross-entropy*), sloužící pro binární problémy, kdy na výstupu očekáváme třídy s hodnotami 0 nebo 1. Na obrázcích 31 až 33 je možné pozorovat graf průběhu učení navržených sítí.



Obrázek 31: Průběh učení první sítě.



Obrázek 32: Průběh učení druhé sítě.

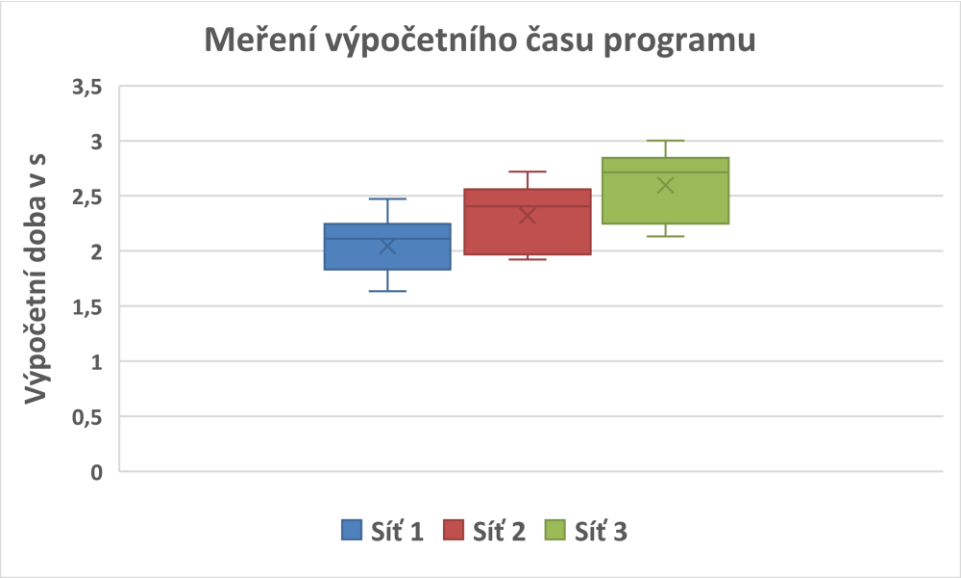


Obrázek 33: Průběh učení třetí sítě.

Na obrázcích 31 až 33 je možné pozorovat, že ve všech případech dochází k určitě míře přeučení. To však může být zapříčiněno rozmanitostí třídy *other*.

Takto natrénované sítě byly dále použity jako konečný prvek programu. První část programu slouží k předzpracování obrazu, vychází z metody pospané v kapitole 6.1.2. Objekty, které spadají do barevného rozmezí, jsou rozšířeny o 10 obrazových bodů na osách x a y, za účelem zachování okolí objektu. Takto připravené objekty, jejichž rozlišení je sjednoceno na 28*28 obrazových bodů, jsou předány na vstup neuronové sítě a klasifikovány do příslušných tříd. Objetky spadající do třídy *varroa* jsou dále spočítány a vykresleny na vstupní obraz.

Sítě byly validovány dvaceti snímky, které nebyly použity pro získání vzorů do datasetu. Do výsledného zhodnocení úspěšnosti sítí bylo přistoupeno i z pohledu výpočetního času potřebného k získání výsledku. Měření bylo provedeno napříč celým programem, od načtení snímku po konečné spočítání celkového počtu výskytu roztoče. Z důvodů neobjektivity, z hlediska simultánně běžících programů systému, bylo provedeno šedesátkrát pro každou síť. Tři údaje pro jeden validační snímek. Z těchto získaných hodnot byl vytvořen graf (obrázek 34) a vypočítán aritmetický průměr pro porovnávací tabulku 8.



Obrázek 34: Graf měření výpočetního času k získání výsledku.

Jak je vidět v tabulce 8, mezi navrženými architekturami dosáhla nejlepšího výsledku síť číslo 2. Předním měřítkem byla validační přesnost, která udává po jaký čas, napříč validační množinou, byla třída *varroa* správně klasifikována. Dále je možné pozorovat, že přidáváním dalších vrstev se validační přesnost zhoršuje a zároveň narůstá i trénovací a testovací chyba. Zdůvodněním může být snížení rozlišovací schopnosti sítě pro síť navrženou s menším počtem vstupních neuronů a větší prohlubování se tedy jeví jako neužitečné.

	Trénovací chyba	**Testovací chyba**	**Validační přesnost**	**Výpočetní čas**
Síť 1	0.0485	0.0633	98.1%	2.04s
Síť 2	0.0379	0.0457	98.9%	2.32s
Síť 3	0.0568	0.1204	94%	2.60s

Tabulka 8: Srovnání výsledků jednotlivých sítí.

Ze získaných výsledků je zřetelné, že se pomocí vhodné kombinace klasických metod a neuronových sítí dají roztoči detekovat s přijatelnou přesností a i malým výpočetním časem.

7 Závěr

Hlavním cílem práce bylo prozkoumat a otestovat metody strojového zpracování a vyhodnocení snímku spadu na podložku v úlu, konkrétně detekci roztoče a počet jeho výskytu na snímku.

V oblasti klasických metod strojového vidění se ideální implemetaci nepodařilo najít, hlavní příčinou byla různorodost pořizovaných snímků. Byly prozkoumány některé funkce knihovny OpenCV, založené na detekci specifického tvaru roztoče a jeho barvě, a bylo experimentováno s nastavením jejich parametrů. Detekce založena na tvaru roztoče vykazuje horší adaptaci při snímcích pořízených v různé vzdálenosti od podložky a nedokáže do množiny detekovaných roztočů správně zařadit všechny roztoče na podložce. U detekce založené na barvě nastává problém s některými nechtěnými objekty, které jsou díky jejich barvě také zařazeny do množiny detekovaných roztočů. Z těchto důvodů bylo k této metodě přistoupeno z pohledu použitelnosti pro předzpracování snímku a extrahování roztočů pro následný vstup neuronové sítě. Dále tato metoda vykazuje méně úspěšné výsledky při pořízení snímků za horších světelných podmínek. Námětem na další práci v této oblasti tedy může být specifikace vzdálenosti pořizovacího zařízení od podložky, která by zamezila široké různorodosti pořizovacích snímků, či případné zkoumání metod, které potlačují horší světelné podmínky.

V oblasti neuronových sítí se podařilo dosáhnout kvalitního výsledku při kombinaci metody založené na barvě roztoče a navržené neuronové sítě o menším počtu vrstev. Toto řešení vykazuje přesnost, která je pro daný problém dostatečná a to za nízkého výpočetního času. Bylo zkoumáno i řešení založené na předučené síti, to se však pro značnou výpočetní / časovou náročnost, s ohledem použití implementace pro mobilní aplikaci, nejevilo jako přijatelné řešení. Při další práci by tedy mohlo být zkoumáno použití neuronových sítí pro celý proces detekce s ohledem na nižší výpočetní / časovou náročnost a experimentování s hlubšími sítěmi.

Přidaná hodnota bakalářské práce v podobě nových znalostí, dovedností a užití bude transferována do praxe formou komerční aplikace pro mobilní operační systém Android, která bude sloužit i jako výstup pro univerzitu. Cílová skupina uživatelů jsou zájmová sdružení včelařů a jednotliví včelaři.

Seznam literatury

[1] ČERMÁK, K. a kol. Včelařství. České Budějovice: PSNV, 2016. ISBN 978-80-260- 9090-8.

[2] BIENEFELD, K. Včelařství krok za krokem. 2. vyd. Líbeznice: Víkend, 2010. ISBN 978-80-7433-023-0.

[3] HLAVÁČ, Václav a Milan ŠONKA. Počítačové vidění. Praha: Grada, 1992. ISBN 80-85424-67-3.

[4] ŽÁRA, J. Moderní počítačová grafika. 2., přeprac. a rozš. vyd. Praha: Computer Press, 2004, s. 542-546. ISBN 80-251-0454-0.

[5] SHAPIRO, LINGA G., and George C. STOCKMAN. Computer vision. Upper Saddle River, NJ: Prentice Hall, 2001. ISBN 978-0130307965.

[6] PETROU M., BOSDOGIANNI P.: Image Processing – The Fundamentals. John Wiley & Sons, New York, 1999. ISBN 0-471-99883-4.

[7] OpenCV team, OpenCV library [online]. 2019 [cit. 2019-03-24]. Dostupné z: http://opencv.org/.

[8] FUMO, David. A Gentle Introduction To Neural Networks Series - Part 1. Towards Data Science [online]. 2017 [cit. 2019-03-25]. Dostupné z: https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-seriespart-1-2b90b87795bc.

[9] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. Deep Learning [online]. MIT Press, 2016 [cit. 2019-03-25]. Dostupné z: http://www.deeplearningbook.org.

[10] Can, A.: Layers of a Convolutional Neural Network. [online]. 2017 [cit. 2019-04-02]. Dostupné z: https://wiki.tum.de/display/lfdv/Layers+of+a+Convolutional+Neural+Network/#LayersofaConvolutionalNeuralNetwork-PoolingLayer.

[11] Specify Layers of Convolutional Neural Network [online]. 2017 [cit. 2019-04-02]. Dostupné z: https://www.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html.

[12] LECUN, Y., B. BOSER, J. S. DENKER, D. HENDERSON, R. E. HOWARD, W. HUBBARD a L. D. JACKEL. Backpropagation Applied to Handwritten Zip Code Recognition. Neural Computation [online]. 1989, 1(4), 541- 551 [cit. 2019-04-08]. DOI: 10.1162/neco.1989.1.4.541. ISSN 0899-7667. Dostupné z: http://www.mitpressjournals.org/doi/10.1162/neco.1989.1.4.541.

[13] KRIZHEVSKY, Alex, Ilya SUTSKEVER a Geoffrey E. HINTON. ImageNet classification with deep convolutional neural networks. Communications of the ACM [online]. 2017, 60(6), 84-90 [cit. 2019-04-06]. DOI: 10.1145/3065386. ISSN 00010782. Dostupné z: http://dl.acm.org/citation.cfm?doid=3098997.3065386.

[14] SZEGEDY, Christian, WEI LIU, YANGQING JIA, et al. Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) [online]. IEEE, 2015, 2015, s. 1-9 [cit. 2019-04-08]. DOI: 10.1109/CVPR.2015.7298594. ISBN 978-1-4673-6964-0. Dostupné z: http://ieeexplore.ieee.org/document/7298594/.

[15] LIN Min, Qiang CHEN a Shuicheng YAN. Network In Network [online]. 2013 [cit. 2019-04-08]. Dostupné z: https://arxiv.org/abs/1312.4400.

[16] RUSSAKOVSKY, Olga, Jia DENG, Hao SU, et al. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision [online]. 2015, 115(3), 211-252 [cit. 2019-04-08]. DOI: 10.1007/s11263-015-0816-y. ISSN 0920-5691. Dostupné z: http://link.springer.com/10.1007/s11263-015-0816-y.

[17] HE K., ZHANG X.; REN S. aj. Deep Residual Learning for Image Recognition. Prosinec 2015 [cit. 2019-04-08]. Dostupné z: https://arxiv.org/abs/1512.03385.

[18] EDJEELECTRONICS. How To Train an Object Detection Classifier for Multiple Objects Using TensorFlow (GPU) on Windows 10 [online]. Seattle, WA, 2018 [cit. 2019-04-08]. Dostupné z: https://github.com/EdjeElectronics/TensorFlow-Object-DetectionAPI-Tutorial-Train-Multiple-Objects-Windows-10.

[19] MARQUES, O. Practical image and video processing using MATLAB., Hoboken, N.J.: Wiley-IEEE Press, 2011, l, 639 p. ISBN 978-0-470-04815-3.