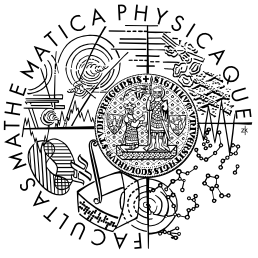


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Josef Bálek

Vyhledání včelí královny v obrazu rámečku

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: doc. Mgr. Martin Pilát, Ph.D.

Studijní program: Informatika (B0613A140006)

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Poděkování

Děkuji doc. Mgr. Martinu Pilátovi, Ph.D. za přátelské vedení mé bakalářské práce a cenné rady, které mě posouvaly kupředu nejen během psaní textu. Děkuji i své rodině a svým blízkým za jejich neustálou podporu a motivaci.

Název práce: Vyhledání včelí královny v obrazu rámečku

Autor: Josef Bálek

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: doc. Mgr. Martin Pilát, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Bakalářská práce se zabývá problémem detekce včelí královny. Při manipulaci se včelami je potřeba královnu rychle najít a označit ji pro následující včelařské úkony. V první části jsou popsány metody strojového učení, které lze využít při trénování modelů. Následně jsou porovnány dostupné modely určené pro tento problém. Po výběru vhodného modelu je popsáno, jak maximalizovat úspěšnost. V praktické části je popsáno vylepšení modelu a jeho dotrénování na datech, která jsem si pro tento problém připravil. Nakonec se model použije pro mobilní aplikaci na otestování v praxi.

Klíčová slova: detekce včelí královny, neuronová síť, mobilní aplikace

Title: Searching for the Queen Bee in an Image of the Beehive Frame

Author: Josef Bálek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: doc. Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The bachelor's thesis deals with the problem of queen bee detection. When handling bees we need to quickly find the queen bee and mark it for subsequent beekeeping operations. The first part describes the methods of machine learning that can be used to train models. Then available models designed for this problem are compared. After choosing a suitable model, it is described how to maximize the success rate. The practical part describes the improvement of the model and its training on the data that I have prepared for the problem. Finally the model will be used for a mobile application for practical testing.

Keywords: queen bee detection, neural network, mobile app

Obsah

Včelařský slovníček

úl, včelstvo - soubor včel s jednou královnou, pozn. úl označuje spíše obydlí, ale někdy se používá v přeneseném významu

podmet - dno úlu

nástavek - jedna vrstva úlu, soubor např. deseti rámečků

rámeček - svisle umístěná vrstva v nástavku, která obsahuje med, pyl, vajíčka, larvy a pohybuje se po ní většina včel

včelí královna, matka - včela, samička, je jediná v úlu, která klade vajíčka

trubec - samec včely, desítky jedinců v úlu

dělnice - samička, která neklade, ale zajišťuje různé činnosti, desítky tisíc v úlu

Úvod

Hledání včelí královny zažil asi každý včelař. V úlu máme desítky tisíc včel a jedná se o dělnice, trubce a matku neboli včelí královnu. Každá včela má své charakteristické rysy, ale matka je přeci jen trochu jiná. Zároveň má ve včelstvu speciální úlohu. Jako jediná zajišťuje reprodukci, a proto s ní musíme zacházet opatrně. Při pravidelné prohlídce včel nás vždy zajímá, v jakém je stavu.

Najít včelí královnu je mnohdy nelehký úkol. Naštěstí se od ostatních včel liší velikostí i vzhledem. Udávat typickou barvu může být zavádějící, protože každá matka má trochu jiné zbarvení. Zbarvení je dáno původem a lokalitou, ale charakteristický rys královny je zvětšený hrudník a protáhlý zadeček. Díky tomuto tvaru ji můžeme rozpoznat mezi dělnicemi. Z dlouhodobého hlediska to je však činnost, kterou je potřeba urychlit, a proto si matku často na hrudníku značíme např. barevnou značkou, fixem, číslem apod. Tím se stane výraznější a při příští prohlídce ji lépe rozpoznáme mezi stovkami včel na jednom rámečku.

Důležité je uvést, že při prohlídce nechceme včelstvo oslabit tím, že snížíme teplotu v úlu. Včely by musely poté ohřát celý prostor a to často vede ke snížení produkce. Zároveň nechceme moc ohrozit královnu samotnou. Té se navíc postupem času vytrácejí orientační dovednosti mimo úl, protože ho od oplodňovacího proletu neopouští. Proto je vypadnutí matky z úlu velkým rizikem.

Naším úkolem je efektivně najít a označit matku již při první prohlídce. Bohužel tím tento úkol nekončí. Ve včelstvu se často stane, že včely vycítí slábnoucí sílu feromonů od královny a vymění ji za novou. Tato výměna může proběhnout vícero způsoby, občas ji může zavinit i sám včelař, ale v každém případě máme najednou novou královnu, která je neznačená a znovu tu máme stejný úkol. Navíc se z různých zdrojů dočteme, že průměrný včelař má okolo deseti včelstev. [1] Potom se z tohoto úkolu stává strojová záležitost, kterou bychom si rádi usnadnili. Zároveň je tento problém svojí podstatou vhodný pro metody strojového učení.

V následujících kapitolách si ukážeme detailnější popis problému i jeho řešení. Existují již různá řešení, ale my se podíváme na pár z nich, která splňují naše požadavky, porovnáme je. Zároveň probereme dostupné metody strojového učení vhodné pro tento problém. Nakonec se pokusíme předložit včelaři nástroj, který mu usnadní život.

Cílem práce je získat trénovací data, otestovat existující modely a vytvořit aplikaci detekující včelí královnu.

1 Analýza a požadavky

V této kapitole si představíme typický problém, který řeší každý včelař. Následně si zadefinujeme požadavky, které máme na software, rozebereme styl použití a očekávaný výstup. Samozřejmě je tento problém dlouhodobý, proto se podíváme na již existující modely a aplikace, které jsou vhodné k použití. Využijeme znalosti existujících modelů pro specifikaci naší aplikace a také při porovnávání modelů.

1.1 Popis problému

Představme si, že máme svých deset včelstev a jdeme za hezkého slunečného odpoledne udělat pravidelnou prohlídku včel. Uvažujme, že máme včely ve dvou nástavcích a matka klade v obou. Otevřeme víko od úlu a začneme ji hledat v horním nástavku. Záleží na typu a rozměrech úlu, ale nástavek má většinou deset rámečků. Prohlédneme každý z deseti rámečků z obou stran. Může se stát, že matku nenajdeme, nástavek odložíme bokem na pevnou podložku a projdeme stejným způsobem i spodní nástavek. Teoreticky můžeme prohlédnout i podmet, ale tam může být velký shluk včel, ve kterém se matka lehko ztratí. Navíc zde nemůže klást, takže to není běžné místo, kde by se vyskytovala. Může se ovšem stát, že tam spadne při manipulaci s rámečky. V nejhorším případě matku nenajdeme. Co s tím?

Můžeme usoudit, že ji nepotřebujeme nutně najít. To je vhodné v případě, když vidíme nakladená vajíčka, takže je její přítomnost v úlu zjevná. Dále si můžeme říct, že nechceme riskovat přimáčknutí matky ke stěně a najdeme ji příště. Ovšem zapálený včelař ji může chtít najít okamžitě.

Ať už se rozhodneme jakkoliv, bylo by nejlepší, kdybychom matku našli již při prvním procházení všech rámečků a při prvním výskytu ji označili. Samozřejmě se vždy může stát, že budeme mít smůlu. Označení dosud neznačené královny je přirozené a usnadní nám práci do budoucna.

1.2 Požadavky

Který nástroj by nám mohl pomoct při hledání včelí královny? Zde se nabízí možnost mobilní aplikace. Chytrý mobil má v dnešní době téměř každý, takže toho můžeme využít. Aplikace bude dobrým prostředníkem, který nám poskytne možnost propojení fotek a modelu pro detekci včelí královny. Nepotřebujeme nikterak složitou aplikaci. Stačí nám nahrát fotku z kamery nebo z galerie a aplikace nám oznámí, jestli se na daném obrázku vyskytuje matka.

Lze namítnout, že dotykový mobil nemá každý, což je pravda zvláště u starších včelařů, u kterých je tento koníček velmi oblíbený. Zde by poté bylo třeba vyrobit zařízení s kamerou a displejem, které by rovnou detekovalo královnu na zachyceném obrázku. Bohužel tento problém je poněkud náročnější, proto se budeme věnovat mobilní aplikaci.

Aplikace by měla být schopna rozpoznat královny od dělnic. Jelikož se včely pohybují celkem rychle, tak máme požadavky na rychlost a samozřejmě na přesnost. Co se týče rychlosti vyhodnocení, tak bychom potřebovali znát odpověď v řádu

nižších jednotek vteřin. Tato délka potřebná na vyhodnocení snímku je pořád nižší, než za jakou dobu detailně prohlédneme celou stranu rámečku pouhým okem. Pokud totiž prohlížíme rámeček po úsecích, je dost možné, že nám to zabere i desítky vteřin. Pokud zvládneme tento požadavek na rychlost, tak máme celkem dobrou šanci, že vyhodnocený obrázek se nebude moc lišit od aktuálního stavu včel na rámečku.

Zbývá nám vyřešit otázku přesnosti. Pokud by nám model vracel predikci na královnu s jistotou vyšší než požadovaný práh *(threshold)* (např. 50%), pak bychom mohli být spokojeni. Zároveň zde probereme, jaké nároky máme na *precision* a *recall*. Pojďme si říct, v jakých případech by aplikace měla označit královnu. Samozřejmě nejlépe by ji měla označit vždy, pokud je na rámečku, ale jsou zde omezení a výjimky. Pokud je královna pod včelami nebo někde u hrany rámečku a není vidět, pak ji ani aplikace neodhalí, protože ji také neuvidí.

V každém případě požadujeme, aby aplikace s nějakou přesností označila vždy jen královnu. Tedy chceme minimalizovat počet falešných výskytů *(false positives)*, jinak řečeno minimalizovat počet případů, kdy aplikace za královnu označí něco jiného. Z tohoto popisu nám vzniká nárok na velkou *precision*. Tato *precision* je poměr správně označených výskytů *(true positives)* ku všem označeným výskytům *(true positives + false positives)*.

Na druhou stranu můžeme slevit v nárocích na *recall* a to z již zmíněných důvodů, kdy matka není vidět. Formálně řečeno poměr správně označených výskytů *(true positives)* ku relevantním výskytům *(true positives + false negatives)* může být nižší než *precision*. *False negatives* jsou výskyty, kdy aplikace matku neoznačí.

Další nároky na aplikaci vznikají se světelnými podmínkami. Ne vždy se nám stane, že fotíme rámečky za slunečného odpoledne. Někdy může být zataženo, jindy můžeme být jen ve stínu a s tím by měl celý dataset pro aplikaci také počítat. Zároveň záleží i na úhlu, pod jakým budeme pořizovat snímky. Pokud by se model učil podle velikosti včel, pak snímek pod jiným úhlem může hodně zkreslit velikosti blízkých a vzdálených včel na rámečku. Toto je také dobré zmínit před trénováním modelu.

1.3 Použití

Zaměřme se na zamýšlené použití aplikace. Pro detekci včelí královny potřebujeme nahrát fotku z galerie, ale především rovnou z naší kamery. Určitě je přínosné mít možnost nahrávat fotky z galerie, pokud by si včelař chtěl dělat statistiky. Tedy při prohlídce včelař nasbírá snímky a poté doma udělá statistiku svých včelstev, např. kolik matek našel na rámečku s plodem, kolik jich bylo u zásob apod.

Druhá varianta bude častější. Vyndáme rámeček z nástavku, nafotíme ho a podle predikce budeme pracovat dál. Určitě by se tento přístup nechal s opatrností využít při vytváření medníku. To je nástavek, kam včely nosí sladinu z okolí a následně se z těchto rámečků získává med. Právě při vytváření tohoto medníku nechceme, aby zde matka kladla, proto se mezi nástavky vkládá mateří mřížka, aby zde královna neprolezla. Detekce její (ne)přítomnosti by nám určitě pomohla, abychom nemuseli sklepávat všechny včely do podmetu a opět riskovat ztrátu matky, ale pořád bude potřeba jisté obezřetnosti. Pokud by se totiž matka dostala do medníku, tak zde naklade a nezbyde zde místo pro med.

Když se zaměříme na podmínky pro připojení k síti, za jakých bude aplikace používána, musíme uvážit lokality, kde včelstva bývají umístěna. Lokality často bývají zahrady u domů a nebo okraje polí, luk a lesů. Možnost nahrání z galerie nám umožňuje i zpětné vyhodnocení, tedy toto vyhodnocení můžeme dělat kdykoliv, pokud na výsledky nespěcháme.

Zásadnější je rychlost u fotky pořízené kamerou, protože u této možnosti potřebujeme odpověď během pár vteřin, jak jsme popsali v části 1.2. Pokud bychom měli model lokálně, tak můžeme vykonávat naši detekci i offline, ale má to omezení na výkon i paměť našeho zařízení. Zároveň by výsledná velikost naší aplikace mohla být relativně vysoká.

Při používání modelu hostovaného na nějakém webu jsme sice limitovaní připojením, ale přináší nám to jisté výhody. Těmi jsou možnost většího modelu a výpočet běžící jinde než na našem zařízení. Pokud očekáváme, že úly stojí na volném prostranství, pak signál bude dostatečný, protože by ho nemělo nic blokovat a velikost pokrytí je velmi vysoká. To je alespoň v České republice velmi pravděpodobné, proto můžeme využít i možnosti hostovaného modelu. Zároveň se v praxi nestává, že by chované včely byly na těžko dostupných místech, kde by nebyl signál, ale je dobré na to pamatovat a zvážit možnosti, s jakým typem modelu pracovat.

1.4 Seznam požadavků

Z předchozího vychází následující seznam požadavků:

- Mobilní aplikace na detekci včelí královny, která nám pomůže s jejím vyhledáváním v úlu.
- Jednoduché ovládání i použití aplikace. Stačí nám nahrát fotku z galerie nebo z kamery a zajímá nás výskyt královny.
- Vysoká úspěšnost aplikace. Zajímá nás především *precision*, případně rozumné F1-skóre, např. 75%. [1]
- Rychlost aplikace. Označení královny očekáváme do pár vteřin.
- Aplikace by měla být dostupná, tedy jednoduše stažitelná.

[1] F1-skóre= $\frac{2PR}{P+R}$, kde P je *precision* a R je *recall*

2 Metody strojového učení a detekce objektů pomocí neuronových sítí

Jak už jsme zmínili v úvodu, opakované vyhledávání včelí královny se stává strojovým úkonem, který musí každý včelař vykonat, když chce matku najít. Hledání matky může být vylepšeno, pokud nebudeme hledat jen okem.[2] Svojí podstatou je tento problém vhodný pro strojové učení *(machine learning)*. Díky tomu, že si můžeme relativně snadno získat data, na kterých určíme, zda se jedná o královnu nebo ne, jde o učení s učitelem *(supervised learning)*, protože k hodnotám máme i očekávané výstupy. Zároveň tento problém spadá do kategorie detekce objektů *(object detection)*. Metody pro detekci objektů se v posledních letech rychle rozvíjejí a díky jejich pestré škále by bylo zajímavé, jak bychom je mohli pro tento problém využít.

Určitě bude zajímavé porovnat různé přístupy a metody strojového učení, zvláště si představíme typy neuronových sítí z oblasti hlubokého učení *(deep learning)*. Právě tyto neuronové sítě jsou vhodné pro detekci objektů a jejich klasifikaci do tříd. Náš problém je v tomto ohledu trochu jednodušší, protože nás zajímá jen jedna třída (královna), ale pořád bude vhodné využít chování neuronových sítí, abychom královnu na obrázku dokázali detekovat. V úvodu jsme popsali odlišný vzhled královny a to bude to podstatné, čeho bychom se při detekci měli držet.

Pojďme se nyní podívat ve zkratce na základy strojového učení, se kterými budeme pracovat dále.

2.1 Data

Pro zjednodušení budeme uvažovat data ve formě vektorů $x = (x_1, x_2, \ldots, x_n)$ a k tomu budeme mít cílovou hodnotu *(target value)* y. Samotným cílem učení je proces, který se snaží předpovídat pro zadaná data takové hodnoty, které se blíží těm reálným. Zároveň očekáváme od modelu, že bude generalizovat, tj. naučí se předpovídat hodnoty i pro dosud nespatřená data.

Teď se podíváme, s jakými daty budeme pracovat. Obrázky jsou matice čísel reprezentující jednotlivé pixely a jejich RGB kanály. Na obrázcích je potřeba mít vyznačené, kde se královna nachází, abychom trénovali model na detekci její pozice. Cílové hodnoty jsou tedy souřadnice její pozice nebo *null*, pokud královna v obrázku není.

Co se týče dat, musíme být opatrní při vytváření datasetu. Ten by měl obsahovat reprezentativní vzorky i vzorky z prostředí, kde budeme model běžně používat. Zároveň bychom se měli vyvarovat tvorbě odlehlých hodnot *(outliers)*, které se výrazně odchylují od zbytku dat. To by náš model začalo negativně ovlivňovat.

Cílové hodnoty mohou být dvojího druhu:

- souřadnice dvou protilehlých rohů pozice: $x_{min}, y_{min}, x_{max}, y_{max}$
  např. Pascal VOC formát
- souřadnice jednoho rohu, výška a šířka objektu: $x, y, width, height$
  např. COCO formát

Oba formáty se využívají v praxi. [3] Pomocí těchto hodnot získáme polohu boxu. Tento box *(bounding box)* využijeme k vyznačení královny při detekci, aby uživatel aplikace hned viděl, kde se hledaná matka nachází.

2.2 Neuronová síť

Už podle názvu je neuronová síť *(neural network)* souborem neuronů. Někdy se označuje také jako umělá neuronová síť *(artificial neural network)*. Stejně jako v mozku jsou neurony základní jednotky celé sítě. Neuron má více vstupů x_i a jeden výstup y, který může být v neuronové síti použit vícekrát.

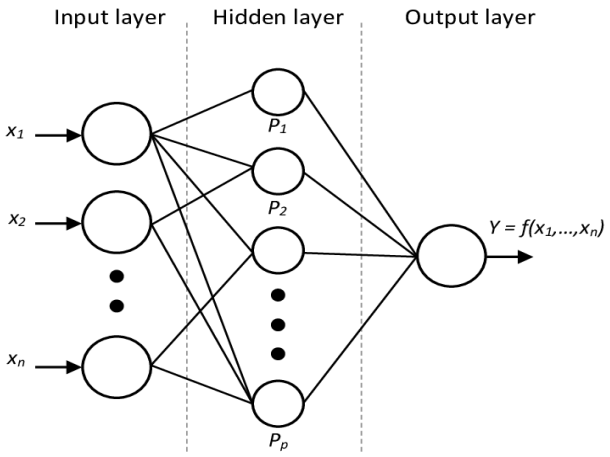
Výpočet neuronu je součtem vážených vstupů:

$$\sum_{i}^{n}(x_i w_i) + b,$$

kde b je práh *(bias)*.

Pojďme se nyní podívat na samotnou strukturu vícevrstvých neuronových sítí. Zaměříme se na jejich zapojení i propagaci parametrů.

V první vrstvě *(input layer)*, je počet neuronů roven dimenzi vstupních dat. Další vrstvy jsou tzv. skryté *(hidden layers)*. Až poslední vrstva je pro nás viditelná, protože z ní máme výstup *(output layer)*. Často tyto vrstvy bývají plně propojené, tedy neuron ve vrstvě L má vstupy ze všech neuronů z vrstvy předchozí, tedy vrstvy $L - 1$.



Obrázek 2.1 Neuronová síť

Nakonec u neuronů už jen záleží na aktivační funkci, která určuje konečný výstup neuronu. Kvůli gradientním metodám, které se používají v dnešní době, nemůžeme používat aktivační funkce založené na funkci signum, protože ta má

derivaci téměř všude nula. Místo toho se používá logistická sigmoida: $f(x) = \frac{1}{1+e^{-\lambda x}}$. Derivace sigmoidy je Gaussova funkce. Častější aktivační funkce je pak funkce RELu *(rectified linear unit)* s definicí $f(x) = max(0,x)$.

Poté se v celé síti zaktualizují váhy podle derivace chybové funkce *(loss function)*. Jako chybová funkce se využívá střední kvadratická chyba MSE *(mean squared error)*. Ta se počítá z cílových hodnot a předpovídaných hodnot:

$$L = MSE = \frac{1}{n}\sum_{i}^{n}(y_i - f(x_i))^2.$$

Někdy se ve výpočtu chyby používá $\frac{1}{2}$ místo $\frac{1}{n}$.

2.3 Hluboké učení

Hluboké učení je jedna z metod strojového učení. Název pochází z anglického *(deep learning)*. Tento název odkazuje na hloubku NN, ta má totiž více vrstev, které mezi sebou různě komunikují. Tento typ sítě se označuje jako (hluboká) dopředná síť *((deep) feedforward network)* nebo někdy také jako vícevrstvý perceptron *(multilayer perceptron - MLP)*. [4]

Dopředná síť [4] označuje NN, která předává vyhodnocené informace ze vstupu x do dalších vrstev. Nakonec se informace dostanou přes celou síť až na konec, kde máme výstup y.

Cílem dopředných sítí je aproximovat nějakou funkci f^*, přesněji řečeno se síť snaží naučit parametry modelu, který bude nejlépe vyhovovat zadaným podmínkám, např. nejvíce minimalizuje ztrátovou funkci. Výsledkem sítě jsou parametry funkce. Poté mapování $y = f(x;\theta)$, kde θ jsou parametry modelu, je nejlepší aproximace funkce.

2.3.1 Zpětné šíření

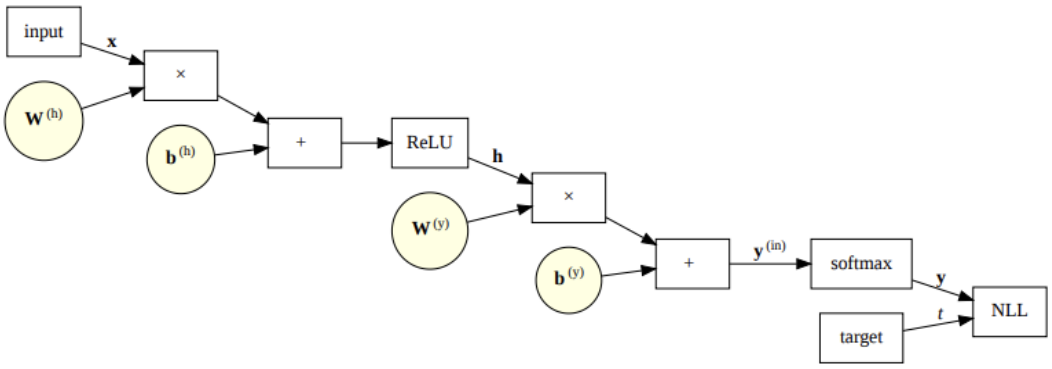
Výsledná funkce f je typicky tvořená skládáním funkcí $f_1, f_2 \dots f_n$ pro hlubokou síť tvořenou n vrstvami. Funkci f lze zapsat jako $f_n(f_{n-1}(\dots(f_2(f_1(x)))\dots))$, kterou aplikujeme na vstup x, f_1 odpovídá první vrstvě, f_2 odpovídá druhé vrstvě atd. až do vrstvy n. Z tohoto popisu vychází myšlenka *deep learningu*.

Pro výpočet zpětné propagace *(backward-propagation)* je užitečné začít pracovat s výpočetním grafem, který lze znázornit jako graf s vektory (popř. tenzory) a operacemi, které na ně aplikujeme. Vektory jsou uzly v grafu a operace mezi uzly jsou orientované hrany. Výpočet přechodů propagace odpovídá zpětné cestě v grafu, kde uzly jsou nahrazeny jejich derivacemi.

Abychom si ukázali výpočet zpětné propagace, uvažme funkce $y = g(x)$ a $z = f(g(x)) = f(y)$. Z matematické analýzy použijeme řetězové pravidlo *(chain rule)*, které říká: $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$. Nechť jsou $\mathbf{x} \in \mathbf{R}^m$ a $\mathbf{y} \in \mathbf{R}^n$, g zobrazí z $\mathbf{R}^m$ do $\mathbf{R}^n$, f mapuje z $\mathbf{R}^n$ do $\mathbf{R}$. Pokud $\mathbf{y} = g(\mathbf{x})$ a $\mathbf{z} = f(\mathbf{y})$, pak

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}.$$

K propagaci gradientu se využívá algoritmus zpětného šíření *(back-propagation algorithm)*, který propaguje chyby od posledních vrstev k těm prvním.

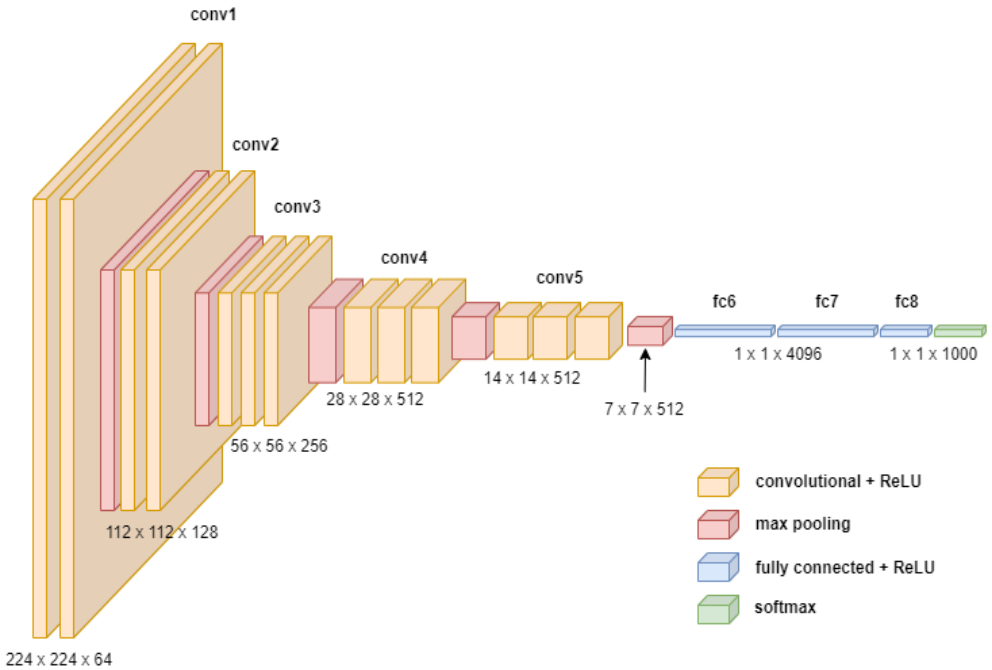


Obrázek 2.2 Obecný algoritmus zpětného šíření, zdroj: [5]

Díky tomuto odvození můžeme říct, že algoritmus zpětného šíření spočívá v provedení součinu gradientů pro každou operaci v grafu. Navíc pokud budeme aplikovat řetězové pravidlo vícekrát, dostaneme všechny potřebné váhy.

2.3.2 Konvoluční sítě

V dnešní době mají dopředné sítě široké využití v oblasti strojového učení. [4] Uveďme například konvoluční neuronové sítě *(convolutional neural network)*, které se využívají právě pro zpracování obrazu. Pokud si zobrazíme aktivace neuronů v jednotlivých vrstvách konvoluční sítě jako obrázky, uvidíme, že na prvních vrstvách po vstupu se chovají jako detektory hran objektů. V hlubších vrstvách sítě mohou jednotlivé neurony detekovat přítomnost složitějších objektů nebo vzorů.



Obrázek 2.3 Konvoluční neuronová síť

Konvoluční sítě čtou jen malé části obrázku (např. 3×3 pixely) a využívají sdílených vah pro všechny pozice. Sdílené váhy v matici se označují jako konvoluční jádro (popř. konvoluční filtr). Toto jádro se posouvá po celém obrázku, čímž

využívá překryvu hodnot. [4] Tím se hodně zredukuje počet vah, které se síť učí. Navíc na obrázek můžeme pustit více konvolučních vrstev, které využívají různě velké konvoluční filtry. Najednou začneme mít více informací o obrázku, ale obrázek při výpočtu potřebujeme i upravovat.

V konvolučních sítích se využívá sub-sampling *(pooling)* vrstvev. Tyto vrstvy čtou také malé části obrázku, ale po přečtení části obrázku vrátí z každého regionu maximum, pokud se jedná o max-pooling. Můžeme si všimnout, že zde se překrývání částí nevyužívá, abychom obrázek redukovali. Následně se oba typy vrstev v konvoluční síti střídají. Pro úplnost uveďme, že na konci sítě je zde pár plně propojených vrstev a následná klasifikace. Klasifikaci provedeme funkcí softmax, které převádí výstupy předchozí vrstvy na rozdělení pravděpodobnosti pro více tříd.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

2.4 Detekce objektů

Detekce objektů *(object detection)* je metoda spojená se zpracováním obrázků a počítačovým viděním, která se snaží najít instance objektů. Tyto objekty mohou spadat do různých tříd. Model určuje ke každému objektu jeho třídu, případně určuje pravděpodobnost dané třídy. V určitých případech může model vracet i pravděpodobnosti např. tří nejpravděpodobnějších tříd. [6]

Detekce se často používá při rozpoznávání tváří nebo rozpoznávání aktivit. V poslední době se *object detection* hojně využívá při vývoji kamer a senzorů pro (polo)autonomní systémy.

Detekce objektů má za sebou dlouhý vývoj a v poslední době zažívá exponenciální nárůst. Tento trend krásně demonstruje počet publikací, které vyšly o *object detection*. [7]

První modely se začaly používat od roku 2000. Jsou to tradiční metody počítačového vidění. Za zmínku stojí Viola-Jones detector a HOG detector. [8]
Viola-Jones používá metodu posuvného okénka *(sliding window)* a různé vylepšení se škálováním a výběrem příznaků. Podobné posuvné okénko se využívá u konvolučních neuronových sítí, jak jsem popsal v části 2.3.2.
HOG *(Histogram of oriented gradients)* detektor [9] je invariantní na škálování. Tuto vlastnost má díky přeškálování vstupu do různých velikostí. Pomocí této techniky má možnost dobře se učit z dat, která obsahují různě velké objekty.

Přibližně od roku 2014 se pro *object detection* začalo využívat hluboké učení. Od té doby se detekce objektů začala vyvíjet velmi rychle. V hlubokém učení existují dva druhy detektorů: dvoustupňové a jednostupňové *(two-stage or one-stage detectors)*.

Jednostupňové detektory zpracují vstup „v jednom kroku“, zatímco dvoustupňové detektory pracují na bázi detekce „od hrubého k jemnému“. Hrubá detekce se snaží zlepšit schopnost vyvolání, zatímco jemná detekce zpřesňuje lokalizaci na základě hrubé detekce a klade větší důraz na schopnost rozlišování.

2.4.1 Dvoustupňové detektory

Příkladem dvoustupňových detektorů je RCNN nebo R-CNN [10] *(Regions with CNN features)* nebo *(Region based CNN)*. Myšlenka RCNN je jednoduchá: model začíná extrakcí sady návrhů objektů (*bounding boxů*) selektivním vyhledáváním. Poté je každý návrh přeškálován na pevný obrázek a vložen do modelu CNN. Tato CNN je předem natrénovaná na ImageNet. ImageNet je velká vizuální databáze pro *object detection*. Nakonec použijeme lineární klasifikátory SVM *(support vector machine)* k predikci přítomnosti objektu v každé oblasti a k rozpoznání kategorií objektů. SVM [7] funguje na maximalizaci vzdálenosti dělící přímky od dvou kategorií. Protože máme typicky více kategorií, ale SVM umí jen binární klasifikaci, musíme udělat kombinovaný klasifikátor.

Pro klasifikaci do více tříd se používají dva typy klasifikátorů:

- „jeden na jednoho“ *(„One-to-One“)*, kde máme klasifikátor na každou dvojici skupin. Předpokládejme n tříd, pak máme $\binom{n}{2}$ klasifikátorů.
- „jeden proti všem“ *(„One-to-Rest“)*, kde máme klasifikátor pro oddělení třídy od zbytku. Při n třídách pak máme n, případně $n - 1$ klasifikátorů.

Po klasifikaci jednotlivých klasifikátorů kombinujeme jejich výstup.

2.4.2 Jednostupňové detektory

U dvoustupňových detektorů můžeme snadno dosáhnout vysoké přesnosti bez jakýchkoli vylepšení, ale zřídka se tyto typy detektorů používají v praxi kvůli nízké rychlosti a vysoké složitosti. Naopak jednostupňové detektory mohou získat všechny objekty v jednokrokovém odvození. [7] Jednostupňových detektorů se využívá pro aplikace provádějící detekci v reálném čase *(real-time detection)*, disponují snadnou implementací, ale jejich výkon je znatelně horší při detekci malých objektů a na snímcích s vysokou hustotou objektů.

Nejznámějším jednostupňovým detektorem je YOLO *(You Only Look Once)* model. YOLO pracuje jinak než dvoustupňové detektory. Používá jedinou neuronovou síť na celý obraz. Tato síť rozděluje obraz do oblastí a předpovídá *bounding boxy* a pravděpodobnosti pro každou oblast současně. I přes velké zlepšení rychlosti detekce YOLO trpí poklesem přesnosti lokalizace ve srovnání s dvoustupňovými detektory, zejména u některých malých objektů. Následující verze YOLO věnovaly tomuto problému větší pozornost. Nedávno byla navržena YOLOv10. [11]

3 Modely

V této kapitole se zaměříme na dostupné modely a metody, které můžeme využít pro náš úkol detekce včelí královny. Zároveň musíme modely umět nějak porovnat. Jedno hodnotící kritérium může být čas strávený detekcí. Další kritérium může být úspěšnost modelu. Ze všech evaluačních metrik, jejichž část jsem popsal v části 1.4 můžeme použít *precision*, *recall* nebo F1-skóre.

Pokud se podíváme na další články, které popisují hodnocení klasifikátorů, [12] narazíme na metriku mAP *(mean Average Precision)*, tedy se jedná o průměrnou *precision* ze všech tříd.

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{|TP_c|}{|TP_c| + |FP_c|}.$$

TP_c označuje počet *true positives* příkladů třídy c a FP_c označuje počet *false positives* příkladů třídy c.

Když teď trochu odbočíme a zaměříme se na úkol, ve kterém chceme klasifikovat jen včelí královnu, tak máme jen jednu třídu a zadaná metrika mAP je jen *precision*. Proto bude možná vhodnější použít F1-skóre, které zahrnuje do ohodnocení *precision* i *recall*.

Zároveň budeme u modelů porovnávat jejich dostupnost, přesněji řečeno možnost použití, dotrénování a zapojení do aplikace. Pro náš úkol potřebujeme buď model, který budeme moct použít lokálně v aplikaci, nebo model někde hostovaný, ke kterému můžeme lehce přistupovat a dotazovat se ho.

3.1 Porovnání modelů

V této sekci se zaměříme na již dostupné modely, které klasifikují včelí královnu. Představíme si různé metody, jak daný problém řešit a také nás bude zajímat efektivita řešení tohoto problému. Zároveň se podíváme na možnosti, které máme pro tvorbu naší aplikace. Tedy projdeme existující koncepty a navržená řešení a pokusíme se o nalezení vhodného způsobu.

3.1.1 Existující aplikace

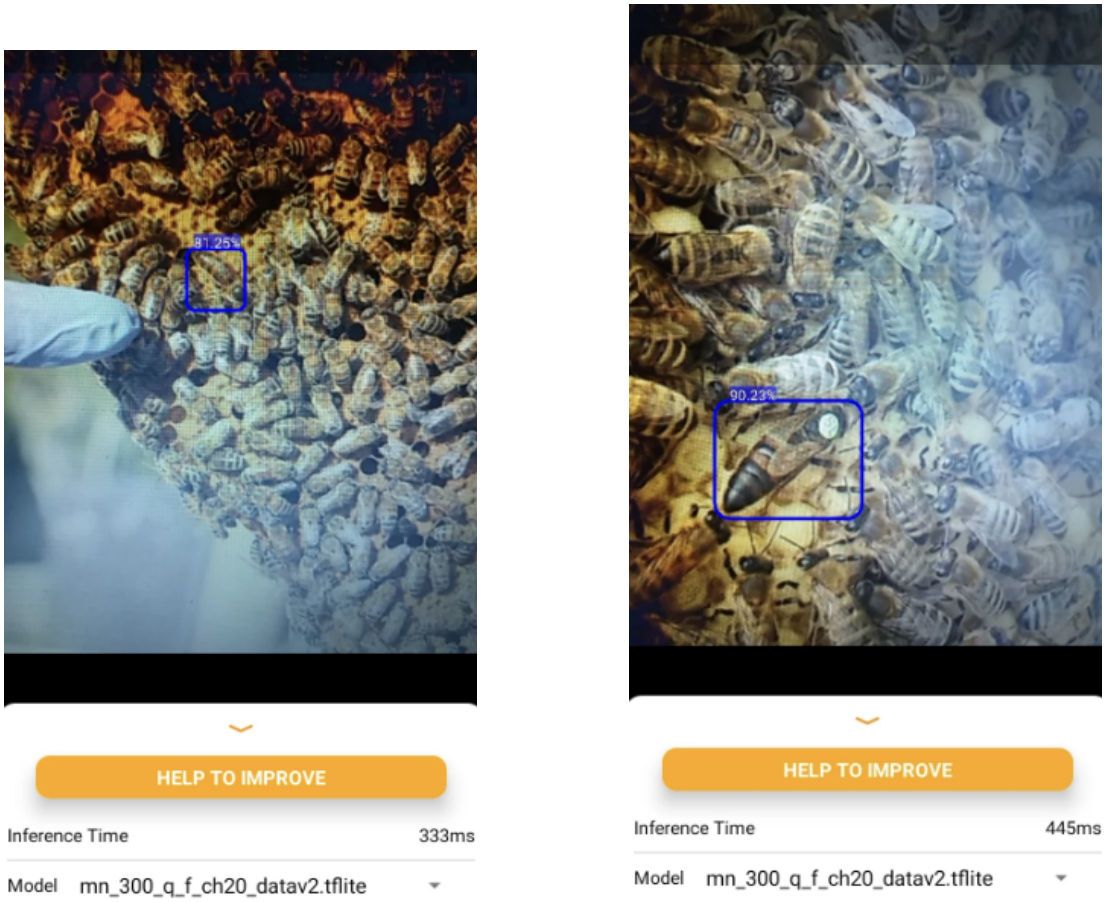
Dříve než začneme vytvářet svoji aplikaci, je dobré si zjistit, co už existuje, abychom se tím mohli případně inspirovat.

Jako první možnost a odrazový můstek se nám nabízí využití již existující aplikace. Když se podíváme do Obchodu Play, uvidíme jednu vyhovující aplikaci. Jedná se o Bee Queen Detector od Pablo AI Team. [13] Tato aplikace částečně splňuje naše požadavky (1.4). Problém s touto aplikací je, že není nikde napsané, jak aplikace přesně pracuje, jak vyhodnocuje snímky a jaké jsou vlastnosti modelu. Zároveň nemá dostupný zdrojový kód, tedy není *open-source*, takže nemůžeme model dotrénovat na našich datech. Když jsem vyzkoušel detekci na svých testovacích datech (./data/test_set), tak měl model v aplikaci F1-skóre 54%. To není špatný výsledek na reálných datech, proto by stačilo model dotrénovat.

Vypozoroval jsem z dat, že matky s barevnou značkou na hrudníku většinou nebyly označeny, stejně tak matky, které neměly typické zbarvení. Předpokládám, že model je natrénovaný na datech, ve kterých je královna dobře viditelná a má typické zbarvení.

Vyhodnocení proběhne do pár vteřin a autor doporučuje nafotit jen třetinu rámečku, aby včely nebyly z veliké vzdálenosti a matka byla o trochu výraznější. Zároveň se doporučuje nehýbat při snímkování mobilním zařízením. Potom detekce funguje uspokojivě. Aplikace vybízí uživatele k podpoře a zaslání dat na další trénování, což je pěkný sběr různorodých dat. Po delším časovém úseku by určitě muselo být zajímavé začít trénovat další model s novými daty.

Příklad *object detection*-aplikace odpovídá popisu, jak uvádí Google Play. Na poskytnutých obrázcích ze vzorového použití aplikace můžeme vidět, že použití je jednoduché: [1]



Obrázek 3.1 Detekce včel externí aplikací, zdroj: [13]

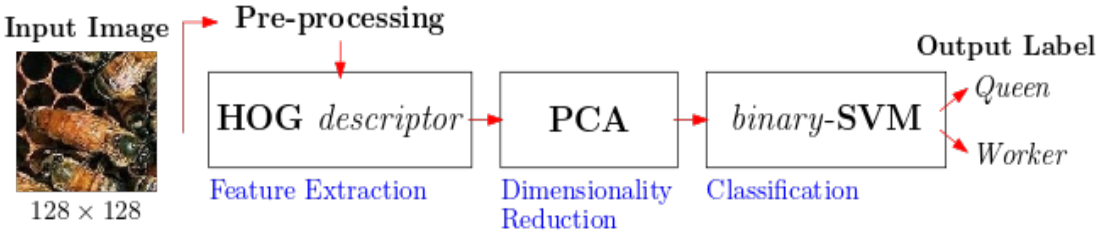
Celkově je aplikace vhodná jako předloha a inspirace pro naši aplikaci. Problém je, že nemá moc detailní popis a nevíme, jaký model interně přesně používá. Předpokládejme tedy, že to bude nějaká lokální R-CNN. Tato komerční aplikace dokonce nabízí možnost výběru typu detekce, např. pomalejší a přesnější nebo rychlá a přibližná detekce. Tato možnost vychází z dostatku dat a zároveň ze softwarových možností společnosti, které dovolují aplikaci výběr typu modelu. Z uživatelského hlediska je velmi přívětivá.

[1] Otestoval jsem chování aplikace na svém zařízení a chování aplikace odpovídá.

3.1.2 Online detektory

Při hledání online modelů jsem našel repozitář na GitHubu od uživatele *vladvroust*, [14] který je dotrénován na detekci včel, ale spíše popisuje obecné koncepty detekce objektů. Problém s tímto zdrojem je, že je relativně starý - 6 let, což při vývoji CNN pro detekci objektů je dlouhá doba. Samotný model může být vhodný, ale za zmíněnou dobu se možnosti posunuly kupředu, proto bude nejlepší, pokud se budeme snažit o co nejnovější typ architektury modelu.

Podívejme se na pěkný model od uživatele *yyaddaden* [15] v externím repozitáři na GitHubu. Uživatel si vytvořil svůj detektor včelích královen. Jedná se spíše o binární klasifikátor, protože tento model bere za vstup malý obrázek včely (128×128 px) a rozhoduje, jestli se jedná o dělnici nebo královnu.



Obrázek 3.2 Navržená struktura modelu, zdroj: [16]

I kdybychom implementačně zvládli využít tento model, tak větší problém je omezenost dat. Těch je pro náš účel určitě málo. Jinak je to dobrý zdroj dat, takže můžeme zapojit do trénování alespoň data z tohoto repozitáře.

3.1.3 Roboflow modely

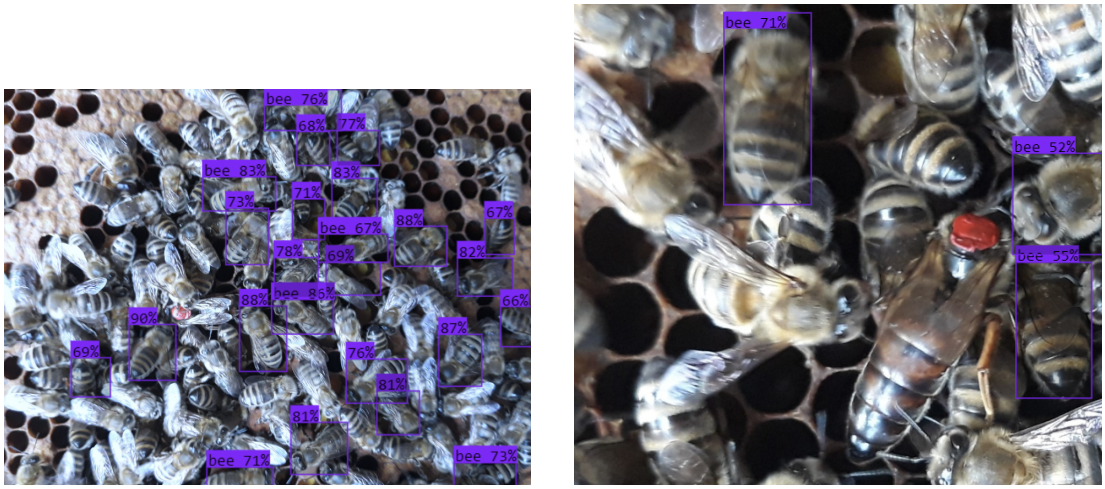
Dalším zdrojem modelů může být web Roboflow, který poskytuje nástroje pro každou fázi procesu počítačového vidění. Na Roboflow jsem našel desítky projektů, které detekují královny, ale často detekují spíše včely obecně.

Online model *Honey Bee Detection Model Dataset* [17] chtěl zaznamenat v reálném čase počty včel s pylem nebo bez něj vstupující do autorova úlu na dvoře. Model využívá YOLOv5 a na klasifikaci pro včely s pylem má F1-skóre 51%. Autor chtěl přidat nějaké další informace k živému přenosu z úlu a aby bylo možné korelovat chování včel při vstupu do úlu s počasím, teplotou nebo jinými podmínkami.

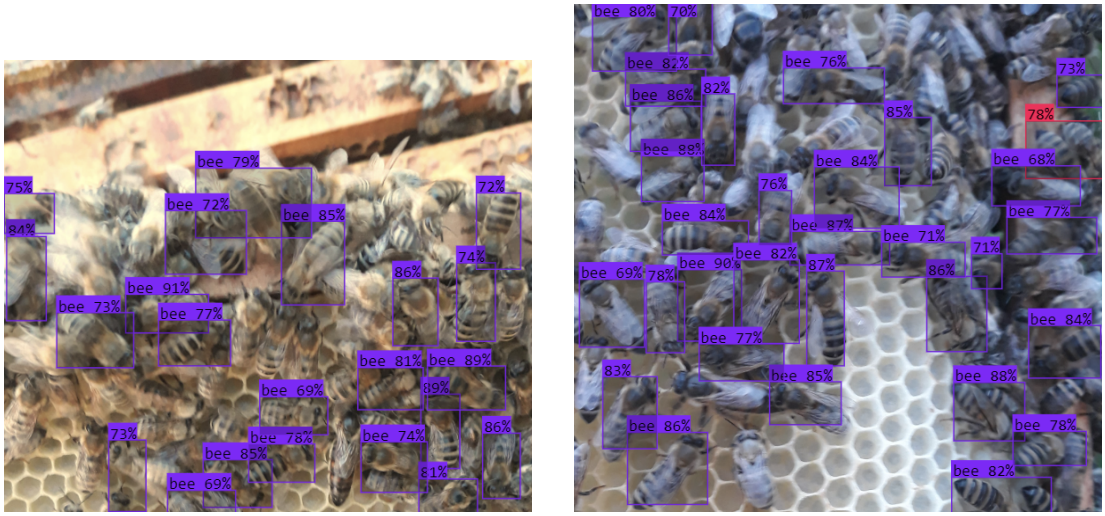
Autor pak přidal další tréninková data, která nejsou specifická pro jeho úl, ale zohledňují klasifikaci trubců a matek kromě včel bez pylu nebo s ním. V současné době model může dobře zachycovat včely s pylem a bez pylu, ale královnu pomocí těchto typů modelů nenajdeme.

Vyzkoušejme model na svých testovacích datech, ve kterých nám půjde o detekci královny. Příklad mého experimentu je na obrázku *3.3*.

Na tomto experimentu můžeme vidět, že královnu model nenašel vůbec. Tedy F1-skóre je 0% pro detekci královny. Vyzkoušel jsem klasifikaci na svých datech a určitě model potřebuje více trénovacích dat. Navíc bychom ze všech včel ani nijak nemohli vybrat královnu např. podle velikosti boxu, protože model není naučen na rozpoznávání královny, proto ji najde zřídka.



(a) značená matka



(b) neznačená matka

Obrázek 3.3 Detekce včel online modelem

3.2 YOLO model

Jak jsme již uvedli v části 2.4.2, YOLO model [18] spadá do jednostupňových detektorů. Tyto detektory přinesly do úloh počítačového vidění velký posun. Jde hlavně o jejich využití v *real-time* úlohách a samozřejmě veliké využití mají i pro statickou detekci.

3.2.1 Historie

Model YOLO byl poprvé představen v roce 2016 v článku *You Only Look Once: Unified, Real-Time Object Detection.* [19] Způsobil převrat v *end-to-end* přístupu k detekci objektů, který umožňoval zpracování v reálném čase. To byl významný pokrok pro komunitu počítačového vidění.

Na rozdíl od předchozích metod je YOLO efektivní architektura detekce objektů vyžadující pouze jednu neuronovou síť. Eliminuje potřebu více běhů nebo dvoukrokového procesu. Tento model dosáhl úžasného mAP 63,4 na datovém souboru PASCAL VOC2007. [19] PASCAL VOC2007 je datová sada, na které se testují různé klasifikační modely, protože obsahuje 20 tříd základních kategorií, patří mezi ně např. osoba, pták, kočka, kolo, auto, motorka, židle, pohovka, atd.

3.2.2 Vývoj a typy YOLO modelů

Joseph Redmon [19] poskytuje stručný přehled o vývoji YOLO modelů pokrývající všechny typy modelů od původního YOLOv1 až po nejnovější YOLOv8. [2]

Metoda detekce objektů YOLO detekuje najednou všechny *bounding boxy* tím, že rozdělí vstupní obrázek na mřížku a předpovídá b *bounding boxů* s danou *confidence* pro c tříd pro každý prvek mřížky.

Každá předpověď *bounding boxu* odráží *confidence* a *accuracy* modelu. Dále obsahuje souřadnice b_x a b_y, což jsou středy rámečku vzhledem k buňce mřížky. B_h a b_w jsou výška a šířka boxu vzhledem k celému obrázku. [3]

Výstupem je tenzor, který může předpovídat více *bounding boxů* s vysokou *confidence*. To zajišťuje rychlou a přesnou detekci objektů.

I když z počátku původní model YOLO byl rychlý a přesný, měl problémy s lokalizací malých objektů a nepřesnými *bounding boxy*. *Predictions* v oblasti *bounding boxu* byly ovlivněny více než jedním objektem a nízká rozlišovací schopnost modelu způsobovala ztrátu jemných detailů.

Modely YOLOv2 a YOLOv3 se snažily řešit původní problémy modelu YOLO. Toho se docílilo zavedením zlepšení v lokalizaci a přesnosti detekce. YOLOv2 zlepšilo celkovou přesnost tím, že využilo techniky, jako je jemnější rozlišení vstupních obrázků, zatímco YOLOv3 zavedlo architekturu s více stupni, která detekuje objekty na různých úrovních měřítka, čímž se zlepšila detekce malých objektů.

YOLOv4 a YOLOv5 pokračovaly v trendu zlepšování přesnosti a rychlosti modelu. YOLOv4 představil nové techniky, jako je CSPDarknet53 pro vylepšení

[2] Existuje i YOLOv9, ale přidané změny jsou již minimální. Také nedávno vyšla YOLOv10, ale nesetkal jsem se zatím v praxi s jejím zapojením.

[3] Formát souřadnic se může lišit, viz. 2.1.

architektury sítě a využití více měřítek pro zvýšení přesnosti detekce. CSPDarknet53 je typ CNN [20], který speciálně zpracovává vstupní příznaky. YOLOv5 model se zaměřil na optimalizaci a snadné nasazení modelu, což vedlo ke zlepšení výkonu na různých platformách a zařízeních.

Vývoj modelů YOLO pokračoval s novými verzemi, které se zaměřují na specifické aplikace a zlepšení výkonu. Každá nová verze přináší vylepšení v přesnosti, rychlosti a schopnosti detekovat různé typy objektů, což dělá YOLO jedním z nejpopulárnějších a nejvíce používaných modelů pro detekci objektů v reálném čase.

4 Dataset, model a aplikace

V této kapitole detailně projdeme moji práci. Zaměříme se na data, trénování modelu a vývoj aplikace. Pro tuto práci využijeme postřehů a pozorovnání z praxe nejen programátorské, ale i včelařské.

4.1 Příprava a sběr dat

Na začátku jsem musel shromáždit dostatek dat na trénování modelu. Nejlepší by bylo vyfotit královnu v různých polohách na rámečku, dále zachytit různá zbarvení královny, model by měl pracovat i při různé intenzitě osvětlení atd., proto by škála dat měla být opravdu veliká. Model chceme trénovat pro neznačené matky, ale často už jsou značené a mohou na svém hrudníku mít značku. Tato značka odpovídá značení v konkrétním kalendářním roce. [21] Podle mezinárodní soustavy se používá těchto 5 barev: bílá, žlutá, červená, zelená, modrá, v tomto pořadí. Dále můžeme použít i dvojciferné označení.

První data jsem hledal na Google Images, ale našel jsem jich jen pár, protože většina byla nepoužitelná pro trénování modelu. Díval jsem se do projektů Roboflow, ale tam jsem vhodná data nenašel. Měl jsem štěstí při hledání na GitHubu, protože tam jsem našel již zmíněný repozitář [15] (viz část 3.1.2), který obsahuje přes 200 fotek královen. Jedná se o malé fotky (128×128 px), ale včelí královna je na nich nafocena v různých pozicích, při různém osvětlení a obrázky zachycují královny různých zbarvení a morfologických znaků.

Dále jsem se vydal nafotit reálná data. Po několika prohlídkách včel a celkových hodinách hledání neznačené královny se mi povedlo nafotit stovky obrázků s královnou v různých pozicích a také jsem natočil desítky videí, která jsem také použil.

Fotky jsem sbíral na podzim roku 2023 a na jaře roku 2024. Pro větší variabilitu jsem přidával fotky značených matek, ale zachytil jsem především jednu barvu značení (červenou). Nějaké jednotky fotek v datasetu obsahují i zelenou a bílou značku, ale rozhodně to není dostatek a je zde prostor pro případně zlepšení. Fotky jsou k nalezení i na mém fakultním GitLabu. [22]

Po získání fotek jsem provedl jejich augmentaci, abych navýšil objem dat. Jednalo se především o přiblížení a ořezávání obrázků, aby matka byla výraznější. Toto doplnění jsem prováděl v aplikaci Windows Fotografie a dosáhl jsem počtu přibližně 1800 různých obrázků včetně těch stažených. Augmentaci jsem nedělal automaticky, abych měl větší kontrolu nad úpravami.

4.2 Anotace dat

Dalším krokem bylo vytvoření anotací všech dat. Dopředu jsem zjistil, že různé modely a nástroje používají formáty anotací, které jsem popsal v kapitole 2.1.

K tomuto účelu jsem využil webový nástroj Make Sense [23], který je vhodný pro vyznačování a přiřazování více kategorií. MakeSense je *open-source* a zdarma, lze ho používat pod licencí GPLv3 *(General Public Licence)*. Navíc není vyžadována žádná instalace. Nalezneme zde podporu více kategorií a ty můžeme vyznačovat

v obrázcích pomocí bodů a různých útvarů: obdélník, čára a polygon. Také má širokou podporu výstupních formátů souborů jako YOLO, VOC XML, JSON nebo CSV. Zároveň využívá AI pro nápovědu u kategorií.

Po dokončení anotování jsem anotace uložil ve formátu CSV a nalezneme je v mém repozitáři. [22]

4.3 Tvorba aplikace

V této fázi vývoje jsem začal pracovat na samotné aplikaci. [1] Tu jsem vyvíjel v Android Studiu. Jedná se o vývojové prostředí založené na IntelliJ IDEA, které podporuje tvorbu mobilních aplikací ve více jazycích.

Pro svoji tvorbu jsem zvolil Flutter. To je *open-source* uživatelské rozhraní SDK na vývoj softwaru, vytvořené společností Google. Ohledně jazyka jsem zvolil Dart, protože je přístupný, přenosný a produktivní pro vysoce kvalitní aplikace na jakékoli platformě.

Měl jsem v plánu vytvořit TFLite model. TensorFlow je bezplatná knihovna pro strojové učení a všechny úkoly včetně *object detection*. TFLite [24] je verze pro mobilní zařízení, která redukuje velikost modelu a optimalizuje dotazy. Ještě předtím, než jsem začal trénovat model, tak jsem se pokusil o napojení, ale vyskytl se problém. Plugin *tflite_flutter* by měl pomoci s komunikací mezi Dartem a tflite modelem, ale po vyzkoušení různých balíčků a pluginů pro převádění typů modelů nebo pro komunikaci mezi knihovnami jsem zjistil, že nedokážu nijak napojit lokální model. Ve složce ./app/bee_queen_detector je tedy aplikace, která pouze zobrazí obrázek, ale detekci nedokáže udělat kvůli problémům s napojením.

Dalším řešením tedy bylo natrénovat model online a dotazovat se ho. Jak jsem již zmínil při hledání obrázků pro můj dataset, narazil jsem na web Roboflow, který poskytuje takovou službu. Natrénuje libovolný *custom* model a zprostředkuje ho online. Tento způsob jsem vyzkoušel, ale nepovedlo se mi napojit ani online model na aplikaci psanou v Dartu, proto jsem se rozhodl pro napsání aplikace v Kotlinu.

Kotlin je staticky typovaný programovací jazyk běžící nad JVM a v nedávné době se začala používat technologie Kotlin Multiplatform, která je navržena tak, aby zjednodušila vývoj multiplatformních projektů. Zkracuje čas strávený psaním a údržbou stejného kódu pro různé platformy při zachování flexibility a výhod nativního programování. [25] Protože se jedná o novinku a během posledních dvou let se vyvíjí různé knihovny pro práci s Kotlin Multiplatform, rozhodl jsem se aplikaci zatím napsat jen pro Android, ale má potenciál i pro ostatní typy aplikací (iOS, desktop, web).

Následovalo přepsání aplikace do Kotlinu. Výsledek je ve složce ./app/BeeQueenDetector.

4.4 Trénování modelu na Roboflow

Po nahrání na web Roboflow jsem zjistil, že používají vlastní formát anotací, proto jsem začal anotace upravovat do podoby, ze které model anotace přečte. Pro-

[1] Při odkazování na soubory jsou myšleny soubory v repozitáři na GitLabu.

blém nastal při nahrávání upravených anotací. Soubory byly ignorovány. Následně jsem celý proces anotování musel zopakovat.

Web využil i videa, ze kterých získal data a udělal vlastní augmentaci s přepočtem anotací, takže nakonec jsem trénoval na přibližně 4500 obrázcích (./datasets). [22]

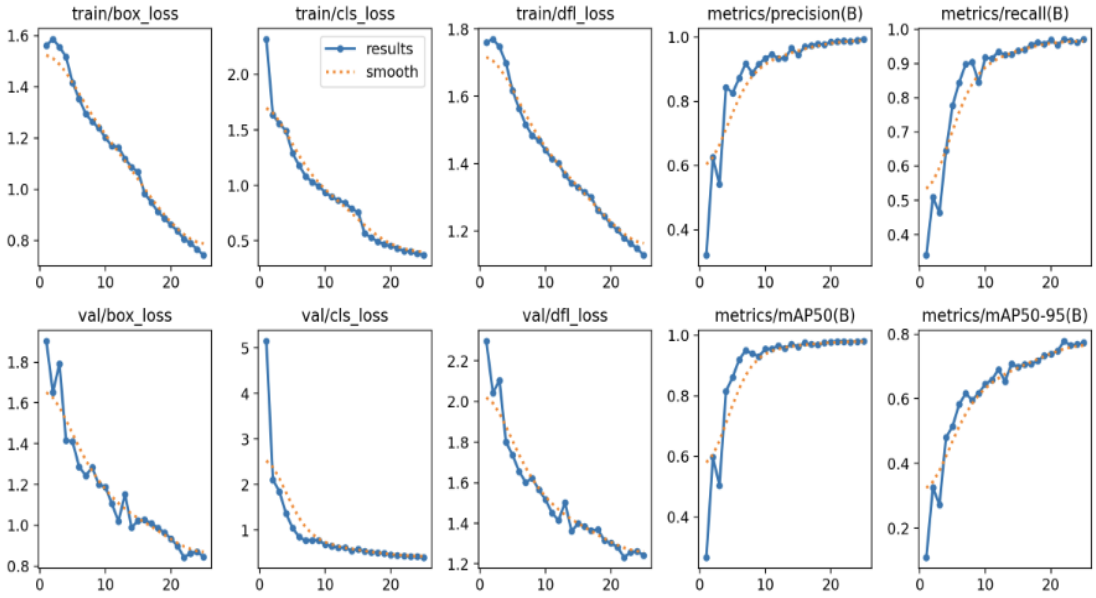
Pro augmentaci byly použity následující parametry:

- rotace: mezi -6° a +6°
- jas: mezi -3 % a +3 %
- rozostření: až 0.5 px
- šum: až 0.14 % px
- výřez: 1 rámeček s rozměry 3 % velikosti obrázku
- *bounding box*-jas: mezi -2 % a +2 %
- *bounding box-shear*: ±1° horizontálně, ±1° vertikálně
- *bounding box-blur*: až 0.1 px

Roboflow 3.0 Object Detection (Fast version) využívá YOLO model pro *object detection*. Aktuálně pracuje na bázi balíčku ultralytics s podporou YOLO modelu verze 8.2. [26] Při detailnějším studování jsem zjistil, že se jedná o typ konvoluční neuronové sítě typu R-CNN.

Začal jsem s trénováním modelu lokálně. Pro tyto účely jsem využil jupyterovského notebooku, který je interaktivní a trénování se na něm dobře pozoruje. Jedná se o ./train-yolov8-object-detection.ipynb. Svůj model jsem trénoval přes dva dny a proběhlo jen 25 epoch.

Zde jsou trénovací křivky:



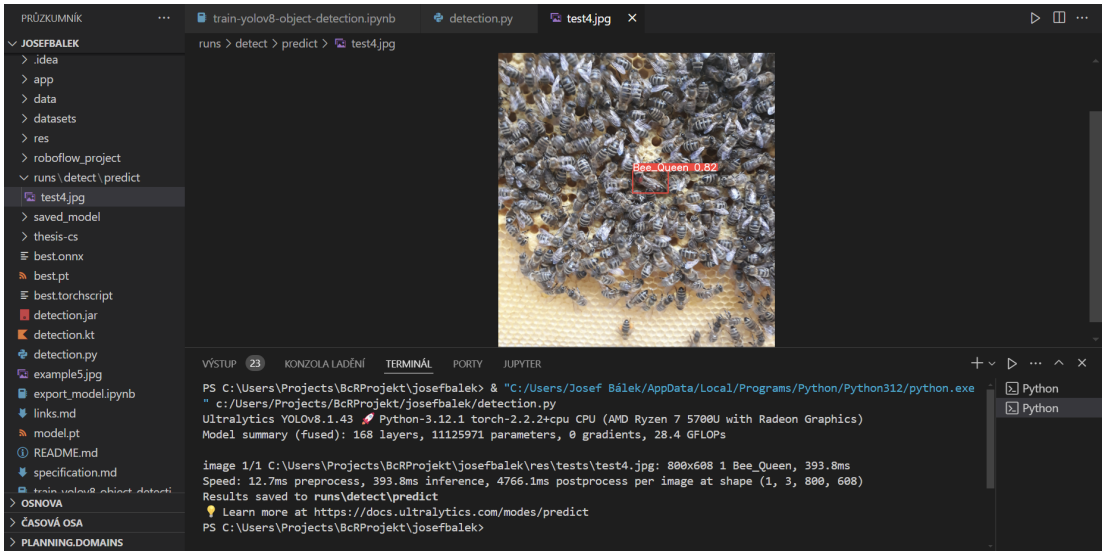
Obrázek 4.1 Trénovací křivky lokálního modelu

Je vidět, že by model bylo potřeba trénovat po další desítky epoch, protože v tomto stavu není úplně dotrénován, což je vidět i z křivek ztrátových funkcí, které ještě nekonvergovaly.

Tento model je z trénovacích běhů uložen jako: ./datasets/yolov8s.pt, jedná se tedy o PyTorch model. Pro další účely jsem udělal kopii nejlepšího modelu: ./best.pt.

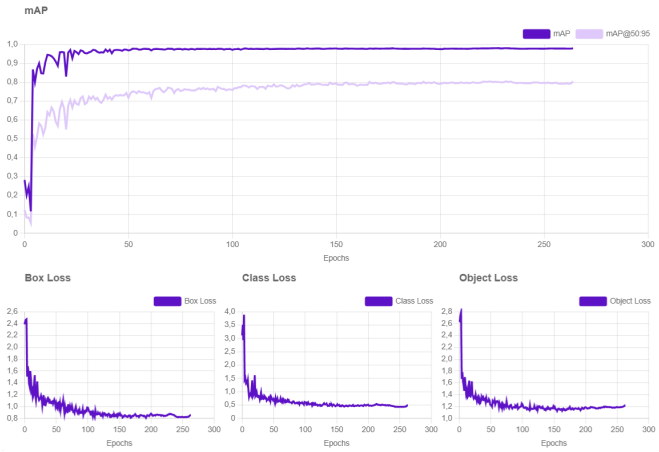
Model lze jednoduše spustit pomocí skriptu ./detection.py, je k tomu zapotřebí pythonních balíčků shutil, subprocess, os, YOLO z ultralytics verze alespoň 8.0 a Image z IPython.display. Pro vyzkoušení doporučuji pracovat v kopii celého repozitáře a výsledek s detekcí nalezneme ve složce ./runs/detect/predict.

Výsledek vypadá takto:



Obrázek 4.2 Detekce lokálního modelu

Lokální model jsem ani díky knihovním funkcím ani díky různým balíčkům nedokázal dostat do podoby, aby s ním mohla aplikace komunikovat. Zkoušel jsem převádět PyTorch model do různých formátů jako jsou onnx nebo torchskript, abych následně dostal model do již zmíněného TFLite modelu, ale nepodařilo se.



Obrázek 4.3 Trénovací křivky online modelu

Model jsem tedy nechal natrénovat i na webu Roboflow. Trénování probíhalo asi 3 hodiny, ale stihlo se natrénovat přes 200 epoch. Pro vyzkoušení hostovaného modelu stačí spustit blok kódu č.8 v jupyterovském notebooku (./train-yolov8-object-detection.ipynb) a je potřeba mít balíček roboflow s modulem Roboflow. Komunikace probíhá díky api-klíči a komunikuje s danou verzí mého modelu. Bezplatný limit pro trénování modelu jsou jen tři pokusy, pro další trénování je potřeba dokoupit kredity, o které lze zdarma zažádat, pokud je člověk pod institucí, která s Roboflow společností spolupracuje. Zatím můžeme bezplatně využívat model do 10000 dotazů měsíčně. Případné navýšení limitu je možné za příplatek. Tento online model jsem napojil do aplikace napsané v Kotlinu.

4.5 Hostovaný model na Roboflow

V této části bych rád ještě popsal výhody a nevýhody, které má vytvořené řešení. Zaměřím se na přesnost modelu i rychlost komunikace a porovnám je s prvotními očekáváními.

Zaměřme se na model jako takový. Model trénovaný lokálně zdaleka nedosahuje takových kvalit, jakých dosahuje hostovaný model, i když záleží na výpočetní síle, kterou máme k dispozici.

Podívejme se nyní na rychlosti obou variant modelů. U lokálního modelu bychom mohli očekávat odpověď v řádu stovek ms až malých jednotek vteřin, záleží jen na rychlosti modelu na daném zařízení. V průběhu měření trvala průměrná délka odpovědi lokálního modelu na notebooku s CPU 580 ms. Na telefonu bude pravděpodobně pomalejší.

U hostovaného modelu může být výhoda, že je k dispozici větší výpočetní síla a detekce proběhně řádově rychleji. Jediné, co nás pozdrží, je *http* spojení a rychlost odpovědi serveru. To bývá vteřina, případně jednotky vteřin, ale záleží na připojení mobilního zařízení. Při měření v domácích podmínkách vycházela průměrná délka odpovědi na 1200 ms.

Dalším aspektem je možnost výměny modelu. U lokálních modelů je nevýhoda, že sice stačí přetrénovat model, ale poté se musí model integrovat do aplikace a pokud bychom chtěli použít jiný formát, tak by to mohlo být náročné. Pokud použijeme stejný formát modelu, tak musíme ještě zajistit, aby se nová aplikace ke všem uživatelům dostala. V tomto ohledu je hostovaný model jednodušší. U něho je nové trénování možné. Pak stačí jen zajistit, že api-klíč a *endpoint* bude odkazovat na komunikaci s novým modelem. V současné aplikaci se nebude muset nic měnit a uživatelé najednou začnou využívat nový model. Zde je vidět jasná výhoda hostovaného modelu, který má jednodušší přechod na nový model.

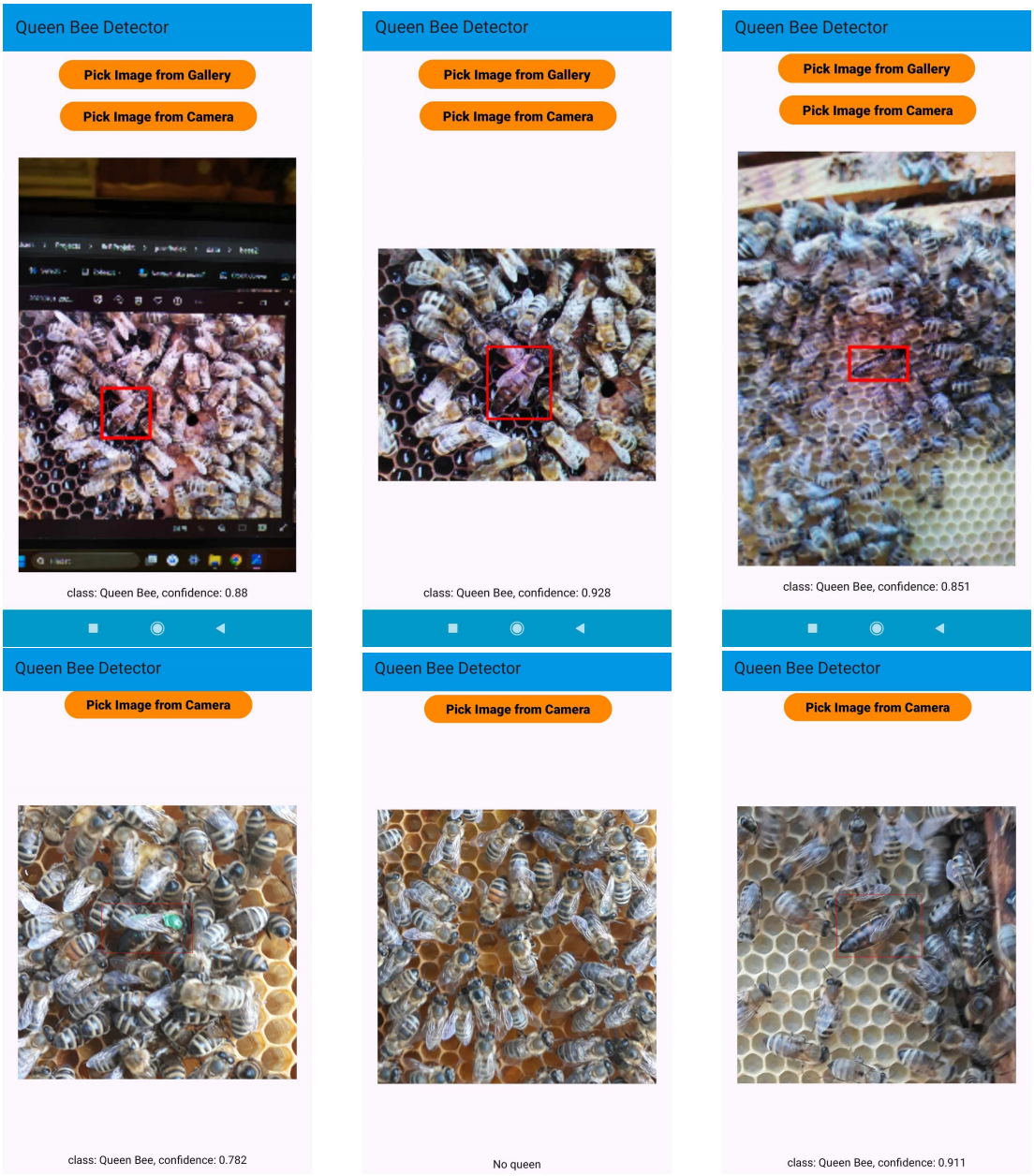
Celkově vidíme, že v mnohých ohledech je hostovaný model nakonec lepší než ten lokální. Je pravda, že lokální může být rychlejší v místech se špatným připojením k internetu, ale na druhou stranu je hostovaný model praktičtější díky tomu, že nezabírá místo v aplikaci a jeho distribuce je o hodně jednodušší. Zároveň je lepší volbou, protože změna modelu a přechod na lepší model je běžná věc.

Nakonec nesmíme zapomenout ani na kvalitu výsledků. Zde jsou oba typy na stejné úrovni. Pokud budeme mít lokální i hostovaný model stejně natrénovaný, budou mít stejnou úspěšnost, ale pro zvýšení úspěšnosti je potřeba dotrénovat model a to je jednodušší u hostovaného modelu, který se dotrénuje na dalších datech a v aplikaci se nemusí nic měnit. U lokálního modelu musíme zapojit i nový

model a následně vydat update aplikace, jinak by se změna u našich uživatelů neprojevila.

4.6 Aplikace

Zde uvedu pár příkladů použití mé aplikace. Můžeme vidět, že po úspěšné detekci nám model vyznačí predikovanou polohu královny a uvede, jak si je klasifikací její třídy jistý. Jsou zde uvedeny příklady fotek, které byly nahrány z galerie. Zároveň jsou zde i detekce na fotkách, které byly přímo vyfoceny. Pokud detekce královnu neodhalí, aplikace místo *confidence* nenález také zahlásí svojí hláškou.



Obrázek 4.4 Detekce královen mojí aplikací

U některých snímků je červený rámeček (*bounding box*) hůře viditelný. Závisí

to na tom, kolik pixelů obsahuje vstupní fotka. V případě, když fotka obsahuje hodně pixelů, je po následném zobrazení čára slabá.

Nakonec jsem provedl měření úspěšnosti modelu. Na zmiňovaných testovacích datech (./data/test_set) 3.1.1 má můj online model F1-skóre 78%.

4.6.1 Uživatelská dokumentace

Aplikace na detekci je velmi jednoduchá a intuitivní. Pro instalaci potřebujeme zařízení s Android 9 (API level 28) nebo vyšší. Po instalaci instalačního souboru [27] není potřeba žádné nastavování a rovnou můžeme aplikaci používat. Nejdříve se nám zobrazí tato úvodní stránka, která nás vybízí pro nahrání fotky:

Obrázek 4.5 Úvodní okno aplikace

Fotku můžeme nahrát dvěma způsoby. Můžeme ji nahrát z galerie. Online detekce nepřijímá moc veliké obrázky. Maximální rozměr obrázku je 1648×1648 px ($\simeq 2.7$ Mpx). Je to z důvodu odesílání fotky na server. Také to slouží jako drobná optimalizace a přenos dat je poté rychlejší, protože jich není tolik. Při překročení maximální povolené velikosti obrázku provedeme automatické zmenšení, které je uživatelsky přívětivější než chybová hláška bez výsledku. Druhá možnost je vyfocení přímo kamerou. V této možnosti se otevře zjednodušené okno fotoaparátu a uživatel může fotit.

Při focení fotky starším telefonem jsem zjistil, že je fotka v horší kvalitě, než je reálně vyfocena. U novějších zařízení jsem tento problém neodhalil. Po odeslání

fotky je na chvíli černé okno, kdy probíhá detekce. Pro uživatele by bylo příjemnější, kdyby se zobrazila nějaká ikona zpracovávání požadavku, ale při dotazování se serveru se vykonává blokující volání, které jsem neuměl obejít. Tím se zabraňuje tomu, aby uživatel nezahltil server více dotazy.

Po provedení detekce se zobrazí obrázek. Podle stavu je na něm vyznačena královna, jak jsem popsal, nebo aplikace oznámí, že královnu nenašla.

Pokud text pod obrázkem není hned viditelný, tak uživatel musí scrollovat dolů. Je to z důvodu responzivity a různé velikosti vkládaných obrázků. Zároveň pozici textu ovlivňuje i velikost obrazovky, tedy u větších telefonů je text viditelný i pod obrázkem bez nutnosti scrollování. Z důvodu lepší přehlednosti výsledku provedené akce se zobrazí krátké oznámení.

Pro případné testování jsou všechny fotky ve složce ./data/. Stejně jako u komerční aplikace 3.1.1 je občas potřeba vyfotit obrázek z větší blízkosti.

4.6.2 Vývojová dokumentace

Při vývoji mého detektoru jsem narazil na implementační problémy 4.3, proto ve složce ./app/bee_queen_detector je jen počátek aplikace, která neumožňuje detekci, a nadále se jí nebudu věnovat. [22]

Zaměřím se nyní na aplikaci v Kotlinu. Ve složce ./app/BeeQueenDetector ji nalezneme se všemi potřebnými soubory. Tato aplikace již úkol detekce zvládá, i když má jistá omezení, která jsem popsal v uživatelské dokumentaci 4.6.1. Hlavní skripty aplikace nalezneme v ./app/BeeQueenDetector/app/src/main/java/com/example/beequendetector. [28] Jedná se o tři skripty v Kotlinu. MainActivity.kt zajišťuje spuštění aplikace, načtení hlavního okna a základní nastavení, které souvisí s předáváním informací o jednotlivých komponentách v hlavním okně.

FirstFragment.kt zajišťuje funkčnost tlačítek příslušnými listenery. Stará se o zobrazování obrázků a pomocí cesty k obrázku a bitmapě obrázku volá metodu doObjectDetection(imagePath: String, bitmap: Bitmap). Hlavní prací této metody je získání a úprava dat z JSONu, který obdrží po provedení detekce. Zpracuje data a vrací polohu nejpravděpodobnějšího objektu. Pokud nemá k dispozici žádná data, vypíše „No queen“. Při úspěšné detekci skript nakreslí daný *bounding box*. Pro úpravu jeho vyznačení stačí upravit *val paint*. Tato proměnná zastupuje štětec, kterým kreslíme v canvas. Nakonec připojíme pod obrázek i *confidence* vyhledané třídy. Víme, že třída bude vždy QueenBee, protože model je tak natrénován. Pro jiný model by toto mohlo být potřeba upravit. Pro upozornění uživatele vždy vyskočí i *Toast* s odpovídající hláškou.

Při provádění metody doObjectDetection voláme ze skriptu Detection.kt třídu Detection s tímto konstruktorem: Detection(apiKey, modelEndpoint, imagePath). Předáváme tedy potřebné informace pro připojení se k modelu a samotný obrázek, resp. cestu k němu. Pokud je obrázek moc velký, automaticky se provede zmenšení i následná detekce. Také vyskočí *Toast* s informací o zmenšení a detekcí královny.

Pomocí HttpURLConnection se spojíme s naším modelem na webu Roboflow pomocí api-klíče a *endpointu* modelu. Tento model jsem detailněji popsal v předcházející části 4.4. Model nám vrátí svoji odpověď a tu vrátíme do našeho skriptu FirstFargment.kt, který výstup zpracuje, jak jsem popsal.

Celkový vzhled aplikace a případné rozložení panelů lze upravit v příslušných xml souborech v ./app/BeeQueenDetector/app/src/main/res/layout. Veškeré

konstanty, používané hodnoty a ikony nalezeneme v ./app/BeeQueenDetector/app/src/main/res.

Aplikace je relativně jednoduchá, ale tím také splňuje naše požadavky, protože nepotřebujeme žádné složité nastavování. Podobného přístupu jsem si všiml i u aplikace Bee Queen Detector od Pablo AI Team. [13]

Pro trénování modelu a následnou detekci jsou v repozitáři další skripty. [22] Jupyterovský notebook ./train-yolov8-object-detection.ipynb zachycuje trénování modelu a následně skript detection.py provádí detekci královny v zadaném obrázku. Detailnější popis přiložených souborů je v README.md. [22]

Závěr

Předmětem této práce bylo vytvoření aplikace pro vyhledávání včelí královny. Nejprve jsme si představili tento problém a probrali jsme, jaké jsou možnosti řešení této úlohy. Přidali jsme poznatky z praxe a uvedli, jaké nástroje bychom mohli využít. Následně jsme si definovali požadavky na aplikaci, která bude schopná vyhledat včelí královnu a zásadně tak urychlí její nalezení v úlu. Tato aplikace by kvůli povaze řešené úlohy měla využívat neuronové sítě pro *object detection* v obrázku.

Po bližším studování různých typů neuronových sítí, jsme zjistili, že pro tuto úlohu se využívají (region-based) konvoluční neuronové sítě. Ty spadají do dvoustupňových detektorů, které jsou sice přesné, ale jsou často velmi složité a pomalé, proto se úplně nehodí pro naši aplikaci. Proto jsme využili jednostupňového detektoru YOLO. Tento detektor se hodně vyvíjí, proto jsme využili jednu z posledních verzí. Dodám, že YOLOv8.2 využívá také R-CNN, ale zároveň i spoustu dalších vylepšení pro rychlejší zpracování.

Porovnali jsme existují aplikaci, dostupné modely a zároveň jsme hledali způsoby, jak si dotrénovat tento detektor na našich datech, která jsme nasbírali, protože potřebujeme, aby na nich vracel uspokojivé výsledky.

Reálná data jsme získali ve včelstvech s neznačenou matkou, ale zároveň jsme sbírali i data se značenou matkou. Následně jsme všechna data anotovali. Tato data jsme využili při trénování našeho modelu. Takto kompletní dataset může dobře posloužit i pro budoucí modely.

Výsledkem je mobilní aplikace pro Android, která na rozumných datech zvládá detekci včelí královny. Tato aplikace má dvě možnosti vstupu. Obrázek můžeme nafotit přímo kamerou nebo ho můžeme nahrát z galerie. Kvůli implementačním problémům naše aplikace komunikuje s hostovaným modelem, ale rozebrali jsme si výhody a nevýhody oproti lokálnímu modelu a zjistili jsme, že to ve většině případů nevadí.

Možnosti rozšíření a vylepšení práce

Aplikace je napsaná v Kotlinu, proto lze aplikaci jednoduše rozšiřovat, např. pokud bychom chtěli zkoušet více modelů nebo chtěli udělat aplikaci multiplatformní. Dále bychom mohli interně upravit práci s fotografiemi, aby uživatel nebyl nijak omezen.

Další možné vylepšení aplikace je, že bychom vytvořili o hodně větší dataset, který by lépe pokrýval spektrum matek, které budeme potkávat v úlech. Mohla by to být data královen se všemi barevnými značkami a následně i královny bez značek za různějších světelných podmínek. Následně bychom natrénovali lepší model, který by byl určitě univerzálnější.

Literatura

1. ČSV. *Český svaz včelařů - informace* [online]. [cit. 2024-07-12]. Dostupné z: https://www.vcelarstvi.cz/cesky-svaz-vcelaru-informace/.
2. Marquis, Marie-Pier; Yaddaden, Yacine; Adda, Mehdi; Gingras, Guillaume; Corriveau-Ctôé, Michael. Automatic Honey Bee Queen Presence Detection on Beehive Frames Using Machine Learning. In: Mahyuddin, Nor Muzlifah; Mat Noor, Nor Rizuan; Mat Sakim, Harsa Amylia (ed.). *Proceedings of the 11th International Conference on Robotics, Vision, Signal Processing and Power Applications*. Singapore: Springer Singapore, 2022, s. 820–826. isbn 978-981-16-8129-5.
3. Singh, Rajdeep. *A Quick Reference for Bounding Boxes in Object Detection* [online]. [cit. 2024-04-27]. Dostupné z: https://medium.com/@rajdeepsingh/a-quick-reference-for-bounding-boxes-in-object-detection-f02119ddb76b.
4. Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
5. Libovický, Jindřich; Straka, Milan (ed.). *Machine Learning for Greenhorns* [online]. [cit. 2024-06-28]. Dostupné z: https://ufal.mff.cuni.cz/~courses/npfl129/2324/slides/?05#3.
6. Dasiopoulou, S.; Mezaris, V.; Kompatsiaris, I.; Papastathis, V.-K.; Strintzis, M.G. Knowledge-assisted semantic video object detection. *IEEE Transactions on Circuits and Systems for Video Technology*. 2005, roč. 15, č. 10, s. 1210–1224. Dostupné z doi: 10.1109/TCSVT.2005.854238.
7. Zou, Zhengxia; Chen, Keyan; Shi, Zhenwei; Guo, Yuhong; Ye, Jieping. Object Detection in 20 Years: A Survey. *Proceedings of the IEEE*. 2023, roč. 111, č. 3, s. 257–276. Dostupné z doi: 10.1109/JPROC.2023.3238524.
8. Giffard, P.V. *Viola-Jones Object Detection Framework*. Tort, 2011. isbn 9786137849767. Dostupné také z: https://books.google.cz/books?id=UZMIywAACAAJ.
9. Hazgui, Mohamed; Ghazouani, Haythem; Barhoumi, Walid. Data Augmentation for Genetic Programming-Driven Late Merging of HOG and Uniform LBP Features for Texture Classification. *Vietnam Journal of Computer Science*. 2024, roč. 11, č. 02, s. 211–239. Dostupné z doi: 10.1142/S2196888823500197.
10. Ren, Shaoqing; He, Kaiming; Girshick, Ross; Sun, Jian. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. Dostupné z arXiv: 1506.01497 [cs.CV].
11. Wang, Ao; Chen, Hui; Liu, Lihao; Chen, Kai; Lin, Zijia; Han, Jungong; Ding, Guiguang. *YOLOv10: Real-Time End-to-End Object Detection*. 2024. Dostupné z arXiv: 2405.14458 [cs.CV].

12. KAUR, Ravpreet; SINGH, Sarbjeet. A comprehensive review of object detection with deep learning. *Digital Signal Processing*. 2023, roč. 132, s. 103812. ISSN 1051-2004. Dostupné z DOI: https://doi.org/10.1016/j.dsp.2022.103812.

13. TEAM, Pablo AI. *Bee Queen Detector* [online]. [cit. 2024-07-13]. Dostupné z: https://play.google.com/store/apps/details?id=org.bqd.queen_detector&hl=en_US&pli=1.

14. VLADVROUST. *Bee Yolo Tracking* [online]. [cit. 2024-07-06]. Dostupné z: https://github.com/vladvroust/bee-yolo-tracking?tab=readme-ov-file.

15. YYADDADEN. *QueenBeeDetection* [online]. [cit. 2024-07-06]. Dostupné z: https://github.com/yyaddaden/QueenBeeDetection/blob/main/README.md.

16. MARQUIS, Marie-Pier; YADDADEN, Yacine; ADDA, Mehdi; GINGRAS, Guillaume; CORRIVEAU-CTÔÉ, Michael. Automatic Honey Bee Queen Presence Detection on Beehive Frames Using Machine Learning. In: MAHYUDDIN, Nor Muzlifah; MAT NOOR, Nor Rizuan; MAT SAKIM, Harsa Amylia (ed.). *Proceedings of the 11th International Conference on Robotics, Vision, Signal Processing and Power Applications*. Singapore: Springer Singapore, 2022, s. 820–826. ISBN 978-981-16-8129-5. Dostupné z DOI: 10.1007/978-981-16-8129-5_125.

17. NUDI, Matt. *Honey Bee Detection Model Dataset* [https://universe.roboflow.com/matt-nudi/honey-bee-detection-model-zgjnb]. Roboflow, 2022. Dostupné také z: https://universe.roboflow.com/matt-nudi/honey-bee-detection-model-zgjnb. visited on 2024-07-13.

18. MODELBIT. *YOLOv8 Model Guide* [online]. [cit. 2024-07-13]. Dostupné z: https://www.modelbit.com/model-hub/yolo-v8-model-guide.

19. REDMON, Joseph; DIVVALA, Santosh; GIRSHICK, Ross; FARHADI, Ali. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. Dostupné z arXiv: 1506.02640 [cs.CV].

20. BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. Dostupné z arXiv: 2004.10934 [cs.CV].

21. JAHAN. *Barva na značení včelích matek* [online]. [cit. 2024-07-09]. Dostupné z: https://www.jahan.cz/barva-na-znaceni-vcelich-matek.

22. BÁLEK, Josef. *GitLab repozitář, Detekce včelí královny* [online]. [cit. 2024-04-26]. Dostupné z: https://gitlab.mff.cuni.cz/teaching/nprg045/pilat/josef-balek.

23. *Make Sense* [online]. [cit. 2024-07-12]. Dostupné z: https://www.makesense.ai/.

24. INC., Ultralytics. *A Guide on YOLOv8 Model Export to TFLite for Deployment* [online]. [cit. 2024-07-13]. Dostupné z: https://docs.ultralytics.com/integrations/tflite/.

25. JetBrains. *Kotlin Multiplatform* [online]. [cit. 2024-07-09]. Dostupné z: https://kotlinlang.org/docs/multiplatform.html.

26. Ultralytics. *Ultralytics YOLOv8* [online]. [cit. 2024-07-16]. Dostupné z: https://github.com/ultralytics/ultralytics.

27. Bálek, Josef. *Queen Bee Detector* [online]. [cit. 2024-07-12]. Dostupné z: https://drive.google.com/drive/folders/1e3jIFpd50uCoD6cQbEi5mNWEorP0E6Mn

28. Bálek, Josef. *Queen Bee Detector* [online]. [cit. 2024-07-12]. Dostupné z: https://gitlab.mff.cuni.cz/teaching/nprg045/pilat/josef-balek/-/tree/master/app/BeeQueenDetector/app/src/main/java/com/example/beequendetector?ref_type=heads.

29. Bálek, Josef. *BeeQueenDetector - v4* [online]. [cit. 2024-07-12]. Dostupné z: https://app.roboflow.com/project/beequeendetector-ta7ig/4.

Seznam obrázků

Seznam použitých zkratek

aj. - anglický jazyk, často použito s odpovídajícím, resp. přesnějším anglickým názvem

AI - umělá inteligence, z aj. *artificial inteligence*

RGB - model míchání barev, z aj. *red-green-blue*

OD - detekce objektů, z aj. *object detection*

MLP - vícevrstvý perceptron, z aj. *multilayer perceptron*

NN - neuronová síť, z aj. *neural network*

CNN - konvoluční neuronová síť, z aj. *convolutional neural network*

NLL - negativní logaritmus *likelihood* funkce, z aj. *negative logarithm of the likelihood function*

SVM - model podpůrných vektorů, z aj. *support vector machine*

YOLO - jednostupňový model, z aj. *You Only Look Once*

mAP - metrika pro klasifikátory, založena na průměrné přesnosti, z aj. *mean Average Precision*

R-CNN - typ konvoluční sítě založená na regionech, z aj. *Region based CNN*

SDK - sada nástrojů specifických pro platformu pro vývojáře, z aj. software development kit

px - pixel/pixelů, bod v obrázku, resp. jednotka grafiky

CPU - centrální procesorová jednotka, z aj. *central processing unit*

A Přílohy

A.1 Dataset s anotacemi

Můj vlastní dataset s anotacemi v repozitáři. [22] Jedná se o fotografie, které jsem nasbíral během několika prohlídek včel. Zároveň jsem využil k trénování modelu i pár forografií z internetu. Ve složce ./data/annotations jsou anotace, které jsem vlastnoručně získal pomocí online nástroje MakeSense [23] pro *labelování* dat.

A.2 Aplikace

Vytvořená aplikace pro detekci, primární je aplikace v Kotlinu, která tento úkol zvládá. Repozitář obsahuje i verzi v Dartu z důvodu vývoje projektu. [22]

A.3 Model natrénovaný na Roboflow

Přikládám i trénovací křivky a detailnější popis modelu včetně popisu k použití. [29] Tomuto modelu jsem nahrál svoje data. Zároveň má tento model vlastní anotaci dat, kterou jsem také provedl.

A.4 Instalace

Také je k dispozici instalační soubor [27] aplikace pro mobilní telefony. [1] Po rozkliknutí odkazu [2] se dostaneme na Disk Google a tam nalezneme požadovaný instalační soubor. Po kliknutí na něj se spustí instalace.

[1] Instalace vyzkoušena pro Android, nikoliv iOS. Tam je problém, že bychom museli naši aplikaci nejprve přidat do App Store.

[2] V případě problému instalace je potřeba adresu vložit do jiného prohlížeče v mobilním zařízení.