Mobilní aplikace pro rozpoznávání zvířat v zoologických zahradách

Jiří Daberger

Bakalářská práce
2023

Do tištěné verze zde vložte oficiální zadání práce, **do PDF verze, která se nahrává do IS/STAG vložte zadání bez podpisů!**

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis studenta

ABSTRAKT

Tato bakalářská práce popisuje vývoj mobilní aplikace určené jako nástroj pro rozpoznávání několika druhů zoologických zvířat. Cílem bylo vytvořit aplikaci na mobilní platformu Android za účelem zpříjemnění návštěv zoo a vzdělávání uživatelů aplikace. Hlavním pilířem aplikace je model konvoluční neuronové sítě, který se stará o vyhodnocení obrázku z kamery mobilního telefonu a zjišťuje, zda se na obrázku vyskytuje některé z předem definovaných zvířat. K naprogramování aplikace byl využitý programovací jazyk Kotlin spolu s nový frameworkem Jetpack Compose a pro natrénování modelu sítě se využily knihovny TensorFlow. Výsledná práce se zaobírá testováním a celkovou úspěšností v přesnosti rozpoznávání zvířat, ale také porovnáním existujících nebo podobných řešení.

Klíčová slova: mobilní aplikace, neuronová síť, rozpoznávání zvířat, Jetpack Compose, TensorFlow

ABSTRACT

This bachelor thesis describes development of a mobile application designed as a tool for recognizing several species of zoological animals. The goal was to create an Android application to make zoo visits more pleasant and educative for its users. The main pillar of the application is the convolutional neural network model, which takes care of evaluating the image from the mobile phone camera and finds out whether any of the pre-defined animals are presented in the image. Programming language Kotlin has been used to create the mobile app together with a new Jetpack Compose framework and TensorFlow libraries have been used for the model training. The result of this work deals with testing and overall success in the accuracy of the animal recognition, but also with comparison of existing or similar solutions.

Keywords: mobile application, neural network, animals recognition, Jetpack Compose, TensorFlow

Poděkování, motto a čestné prohlášení, že odevzdaná verze bakalářské práce a verze elektronická, nahraná do IS/STAG jsou totožné ve znění:

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Potřebné poděkování MetaCentru, nezapomenout aktualizovat, kdyby to změnili.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Hledat TODO pro doplnění

- Nastudujte a popište problematiku spojenou s detekcí objektů v obraze.
- Zvolte vhodné technologie a prostředky k implementaci aplikace.
- Navrhněte mobilní aplikaci pro rozpoznávání vybraných zvířat pomocí fotoaparátu na platformě Android.
- Zvolte vhodná zvířata a vytvořte jejich dataset pro rozpoznání v obraze.
- Implementujte vámi navrženou aplikaci.
- Výslednou implementaci vhodně otestujte a popište výsledky.

OBSAH

ÚVOD

Chytré mobilní telefony a jejich aplikace jsou trendem dnešní společnosti a většina z nás, si život bez chytrého mobilního telefonu nedokáže ani představit. Dnešní telefony již neslouží pouze jako prostředek ke komunikaci s našimi přáteli pomocí telefonování a SMS, ale zejména k využívání chytrých aplikací, které mohou udělat náš život jednodušší.

Mobilní telefony dnes obsahují širokou škálu hardwaru, jako je GPS sloužící pro zjištění aktuální polohy, čip NFC, který mimo jiné umožňuje provádět bezkontaktní platby, kvalitní fotoaparát pro zachycení neopakovatelných okamžiků a v hlavní řadě výkonný procesor, který se o všechny operace dokáže postarat. Výše uvedené možnosti ale vyžadují jednu důležitou věc, kterou je správná mobilní aplikace, díky které je možné využít mobilní telefon prakticky k čemukoliv. Takových aplikací existuje celá řada, některé slouží pro zabavení ve volném čase, některé jako důležitá zdravotní pomůcka nemocným lidem, jiné zase jako pomocník při cestování nebo nakupování a v neposlední řadě ty, které se snaží člověka něco přiučit, naučit nebo mu pomoct vyhledat si další informace. Mezi poslední množinu z výčtu aplikací se může zařadit i aplikace vznikající v této bakalářské práci, jelikož se snaží uživateli usnadnit návštěvu zoologické zahrady a udělat ji více interaktivní za pomocí mobilního fotoaparátu a moderních technologií dnešního světa. Aplikací využívající mobilní fotoaparát je nespočet, převážně se ale jedná o aplikace sloužící jako komunikační prostředek s možností sdílení fotografií, jako jsou sociální sítě Instagram nebo Snapchat, popřípadě aplikace, které dokážou pomocí umělé inteligence detekovat tvář uživatele a zaměnit ji za něco jiného, například za hlavu zvířete, popřípadě tvář digitálně zkrášlit.

Výše zmíněná umělá inteligence je vytvořena za pomoci umělých neuronových sítí, které jsou díky své schopnosti učit se, vhodná pro řešení komplikovaných úloh v oblastech, jako je například klasifikace obrazových dat.

Proto je hlavní myšlenka této práce vývoj mobilní aplikace, jakožto prostředek pro detekci zvířat s využitím naučeného modelu konvoluční neuronové sítě za použití vlastního trénovacího datasetu.

V úvodu teoretické části je představeno odvětví počítačového vidění, do kterého spadá nejen základní problematika této práce. Z počítačového vidění přejdeme k pojmu konvoluční neuronové sítě, kde si popíšeme jejich architekturu a samotný proces učení. Seznámíme se s knihovnami TensorFlow a zjistíme reálné možnosti existujících řešení pro rozpoznávání

objektů v obraze. Před praktickou částí se ještě seznámíme s operačním systémem Android a jeho novým frameworkem Jetpack Compose.

Praktická část již obsahuje samotný proces vývoje aplikace a jejího modelu pro rozpoznávání zvířat. Popisuje jednotlivé problémy a jejich řešení, se kterými se autor během vývoje setkal. Obsahuje detailní popis tvorby dvou datasetů a jejich následné použití pro naučení modelu neuronové sítě, ke kterému dopomohla organizace MetaCentrum zrychlením času trénování. Další část je již zaměřena na vývoj nativní mobilní aplikace programovacím jazykem Kotlin s využitím frameworku Jetpack Compose. Ve finále jsou testováním zhodnoceny výsledky a porovnání s již existujícími řešeními podobného problému této práce.

I. TEORETICKÁ ČÁST

1 POČÍTAČOVÉ VIDĚNÍ

Počítačové vidění neboli **Computer Vision** (**CV**) je trendem dnešního světa. Jedná se o oblast strojového učení, která umožňuje počítačům porozumět okolí, kolem kterého se nachází. K tomuto se využívá sofistikovaných algoritmů a neuronových sítí, které jsou hlavním pilířem této technické oblasti, jelikož právě ony jsou schopné se naučit rozpoznávat jednoduché tvary, znaky či komplexní objekty reálného světa.

1.1 Využití CV

Samotné použití počítačového vidění vyžaduje dostatečně kvalitní kameru, která bude plnit funkci „očí počítače“ a výkonný hardware s procesorem, který se bude starat o zpracování obrazu z kamery. Další ne méně důležitou podmínkou je samotný software/aplikace, která bude převádět výsledky do podoby, se kterou se bude moct dále pracovat.

Existuje široká škála možností, ve kterých by se počítačové vidění dalo využít. Mezi hlavní odvětví se zařazuje to, kde je potřeba nějaký druh automatizace, jako jsou například:

- **Autonomní vozidla** – jejich vývoj jde neustále dopředu, a to hlavně díky CV. Bez možnosti automatizované detekce objektů, jako je v tomto případě řada překážek na silnici, chodci, všechny druhy vozidel, dopravní značení nebo samotná cesta by nebylo možné autonomní vozidla vyvinout.
- **Medicína** – v posledních několika letech se můžeme s CV setkat i v lékařském prostředí, kde dopomáhá určovat diagnózy pacientů z RTG a jiných snímků.
- **Bezpečností systémy** – mezi které můžeme zařadit například bezpečností kamery na pracovištích a parkovištích. Ty automaticky snímají jejich okolí, detekují procházející osoby a v případě problémů nebo zločinů je lze snadno vystopovat.
- **Identifikace** – rychlou a bezpečnou formou moderních pracovišť je rozhodně možnost identifikace pracovníků podle jejich obličeje. Na základě správného rozpoznání obličeje má osoba povolený vstup do areálu nebo místnosti, aniž by se musela zdlouhavě potvrzovat například heslem.
- **Robotika** – robotická ramena nebo vozíky využívající řadu senzorů a kamer pro plynulé pohybování v prostoru s interakcí na okolní vlivy podobně jako autonomní vozidla.

1.2 Klasické úkoly počítačového vidění

Z využití CV je patrné, že se především jedná o procesy detekce nebo identifikace, z čehož vyplývá typická úloha CV, kterou je určení, zda se v obrazových datech ukrývá nějaká hodnota/objekt, který chceme nalézt. Pro člověka je to poměrně jednoduchá úloha, ale při velkém počtu dat taky velice časově náročná.

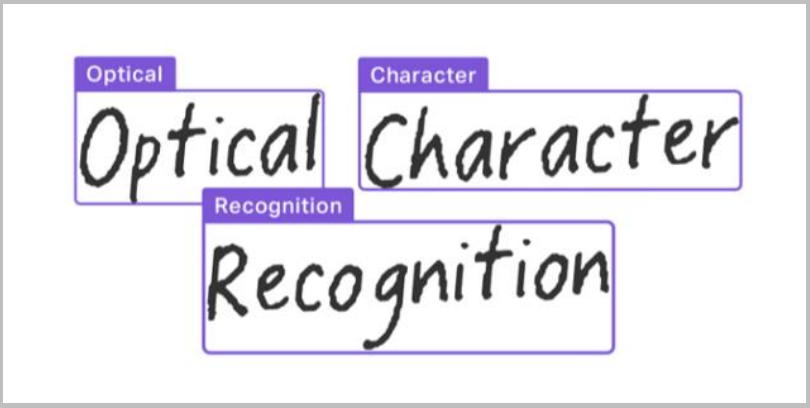
1.2.1 Detekce objektů

Mezi nejznámější úlohu CV se může zařadit tzv. detekce objektů v obraze, která je doprovázena i samotnou klasifikací detekovaného objektu. Detekce spočívá v prohledávání obrazu kousek po kousku s označením pravděpodobnosti výskytu nějakého naučeného vzoru. Finální výsledek obsahuje bodové souřadnice ve 2D obrázku s příslušným názvem objektu. Tímto se detekce objektů liší od **klasifikace obrázku**, ve které je výsledek pouze název vyskytující se třídy na obrázku.

Detekci objektů lze také využít pro jejich sledování či trasování, jako jsou například lidé nebo vozidla ve městech.

1.2.2 OCR

Jedná se o proces digitalizace textu neboli převod ručně psaného, popř. tištěného textu do podoby, se kterou lze s textem pracovat na počítači. V oblasti počítačového vidění spadá OCR pod nejčastěji se vyskytující odvětví. V praxi funguje velice podobně jako detekce objektů, tudíž se hledá oblast v obrázku, která obsahuje větu, slovo nebo znak a jakmile se tato oblast nalezne, je aplikována klasifikace pro rozpoznání konkrétního znaku. [1]



Obrázek 1. Ukázka fungování OCR. [2]

2 KONVOLUČNÍ NEURONOVÉ SÍTĚ

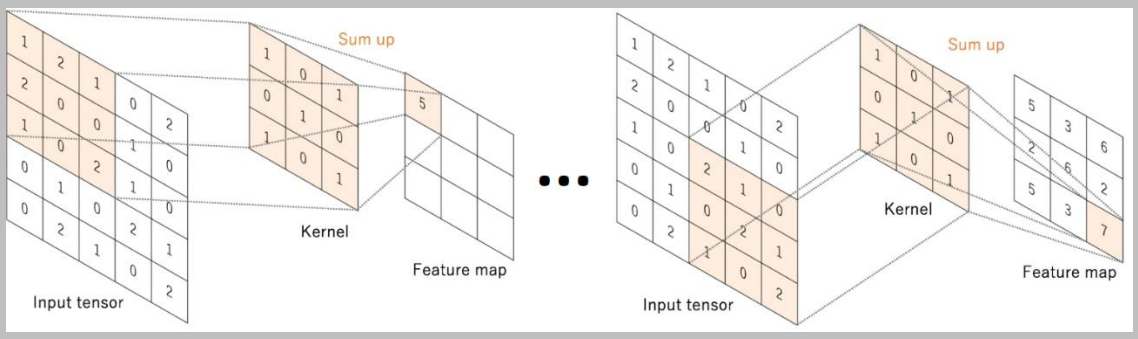
Konvoluční neuronové sítě (anglicky Convolutional Neural Network, **CNN**) jsou nejpoužívanějším typem neuronových sítí starající se o detekci či rozpoznávání objektu v obraze nebo zvuku. Jedná se o rozšířenou variantou klasických neuronových sítí o speciální vrstvu, která dopomáhá elegantně eliminovat problém s velkými daty, jako jsou právě obrázky. Tímto elegantním řešením se zabývá tzv. konvoluční vrstva (od toho poté název Konvoluční neuronové sítě), která značně redukuje vstupní parametry obrazu. [3]

2.1 Vrstvy CNN

Jak již bylo naznačeno, architekturou konvoluční neuronové sítě jsou navzájem propojené vrstvy, které mají každá svá specifika. Tyti vrstvy jsou mezi sebou několikrát střídány, což s různou kombinací zapříčiňuje rozlišné trénování sítě. Proto je návrh po-sobě jdoucích vrstev často nejdůležitější přípravou při trénování sítě.

2.1.1 Konvoluční vrstva

Jedná se o vrstvu, která v určitém směru prochází obraz pomocí několika filtrů, které zachycují různé informace obrázku, jako jsou hrany, světlost, popř. barva a jiné. Tento filtr si můžeme představit jako matici N×N (obvykle 3×3 nebo 5×5) navíc s doplněním barevného kanálu obrázku (×3 pro barevný RGB režim nebo ×1 pro režim šedi). Při procházení, které má navíc nastavení udávající krok (obvykle 1), se generuje tzv. Feature mapa obsahující součet „aktivovaných“ pixelů ze vstupního obrázku a filtru, tak, jak můžeme vidět na obrázku níže (Obrázek 2.)[1]. [4]



Obrázek 2. Aplikace filtru na vstupní obrázek v konvoluční vrstvě. [4]

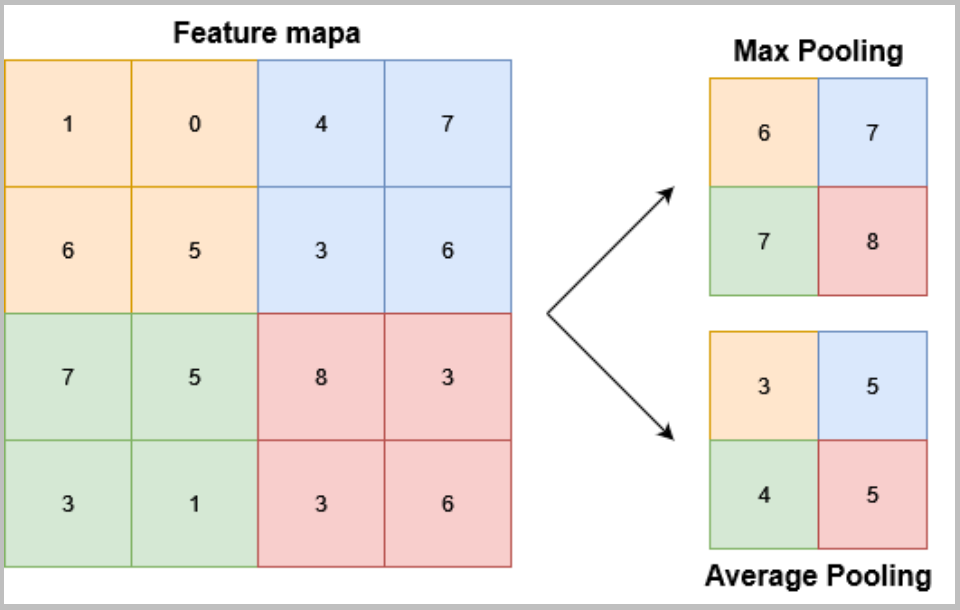
[1] Matici **Kernel** na obrázku chápejme jako **filtr**.

2.1.2 Pooling vrstva

Pooling vrstva neboli vrstva sdružující, se stará o další zredukování vstupních parametrů pomocí jednoho z existujících filtrů. Tato operace se podobá operacím v konvoluční vrstvě, s výjimkou, že vstupní maticí v Pooling vrstvě je výstup z konvoluční vrstvy, tzn. Feature mapa.

Nejčastějším filtrem je operace nazvaná **Max Pooling**, kterou je obvykle matice 2×2 s krokem 2, aby se operace nepřekrývali a tím dosáhli největšího zredukování vstupních parametrů. Tento filtr má za následek to, že se z Feature mapy přečte pole o velikosti 2×2 a do výstupu se z daného pole uloží **maximum**, a tak pokračuje až do konce, vždy s nastaveným krokem 2.

Druhým nejčastějším filtrem je **Average Pooling**, který do výstupu vkládá průměrnou hodnotu z hodnot vstupní matice. [4]



Obrázek 3. Ukázka použití filtrů v Pooling vrstvě.

Z obrázku (Obrázek 3.) je patrné, že se mapa velikostně zredukuje o ¼ díky odstranění redundantních pixelů. To zapříčiní snížení celkového rozlišení obrázku, ale také snížení potřebného výkonu pro další potřebné výpočty a samotné trénování sítě.

2.1.3 Plně propojená a Aktivační vrstva

Doposud se vždy pracovalo s 2-D maticemi, které jsou ale pro plné propojení potřeba vektorizovat neboli převést do 1-D vektoru. Tím se propojí každá vstupní hodnota s výstupní a

vznikne **plně propojená** vrstva (někdy také **dense** vrstva). Typickým počtem výstupních hodnot je počet vstupních tříd, které má daná neuronová síť umět rozpoznat.

V této poslední vrstvě se získává pravděpodobnost každé třídy ze vstupního obrázku. Každá hodnota je v rozsahu 0–1, kde jejich společný součet musí vrátit hodnotu 1. Čím větší číslo (pravděpodobnost) u třídy, tím si je síť více jistá, že se na obrázku nachází daná třída. [4]

2.2 Učení neuronových sítí

Jak již celý koncept umělých neuronových sítí vychází z neuronů lidského těla, ani jejich samotné učení není výjimkou. Jako i my lidé se učíme od jiných lidí, stejně tak se učí i umělé neuronové sítě, kdy se pomocí zpětné vazby dotazují své úspěšnosti trénování.

O trénování sítě se starají dvě hlavní části, které se nazývají **dopředná** a **zpětná** propagace. Každá z těchto „funkcí" se stará o vyhodnocování učení a jeho následnou optimalizaci v dalších iterací učení. Kolikrát se budou iterace opakovat je obsaženo v hodnotě zvané **epoch**. Počet iterací v jedné epoše závisí na 2 hlavních bodech:

1. Počet trénovacích dat
2. Velikost **batch** – jedná se o počet obrázků, na kterých se bude síť učit v jeden moment. Závisí na velikosti grafické paměti a obvyklá hodnota nastavení je 16, 32 nebo 64.

Z následujících dat lze vytvořit rovnici (1), která spočítá počet iterací ***i*** mezi podílem počtu trénovacích dat ***dataSize*** a velikostí ***batch***.

$$i = \frac{dataSize}{batch} \tag{1}$$

1 epocha tedy udává jednu celou iteraci učení přes všechna trénovací data. Z toho lze určit druhá rovnice (2), která udává celkový počet všech iterací ***sumI*** učení sítě, než bude trénování dokončeno.

$$sumI = i * epoch \tag{2}$$

Každá iterace je tedy složena z **dopředné propagace**, která přijímá trénovací data na vstupu a pomocí jednotlivých vah neuronu a očekávaného výstupu rozhoduje o jejich aktivaci. Po této části následuje **zpětná propagace**, ve které se váhy neuronů mírně upravují pro zajištění lepší přesnosti sítě v dalších iterací. Tato úprava je závislá na chybě, která vznikne mezi očekávaným výstupem a skutečným výstupem. [5]

2.2.1 TensorFlow

Jedná se o známou knihovnu používanou pro projekty strojového učení, jako je například analýza dat, klasifikace obrazu nebo překlad textu. Byla vytvořena týmem Google Brain v roce 2015 a volně poskytnuta jako open-source. [6]

TensorFlow funguje na principu toku dat skrze graf. Data jsou v tomto případě reprezentovány N-dimenzionálními strukturami, které jsou nazvané **Tensory**, z čehož vyplývá i samotný název knihovny. Struktura uložených dat je buď vektor nebo matice. **Graf** obsahuje propojené uzly, které jsou využity pro aplikování různých operací na datech (Tensorech). K dosažení své vysoké výpočetní rychlosti bylo TensorFlow vytvořeno za pomocí programovacího jazyka C++, ale důležitou roli hraje Python API pro zajištění snadnější dostupnosti knihovny. [6]

Výpočetní graf je datovou strukturou, jejíž hlavní výhoda je možné uložení, znovu rozběhnutí nebo například i její vizualizace. Z této flexibility těží celá knihovna TensorFlow, jelikož je takto vytvořené grafy možné exportovat na různá zařízení jiných systémů i architektur. [7]

2.2.1.1 TensorFlow Lite

Jelikož se tato práce zabývá vývojem na mobilní platformu, je potřeba zajistit kompatibilní model strojového učení na mobilní zařízení. Jak je známo, mobilní zařízení mají omezené množství výkonu i paměti, díky čemuž by na nich nebylo možné klasický model spustit.

TensorFlow proto přišel s nástrojem **TensorFlow Lite**, který je speciálně navržen pro mobilní a embedded zařízení, jenž eliminuje základní problémy strojového učení na mobilních zařízeních.

- **Prodleva** – není potřeba server pro uložení modelu, jelikož je model uložen na samotném zařízení
- **Soukromí** – data nejsou posílána ze zařízení
- **Konektivita** – internetové připojení není potřebné
- **Velikost** – velikost modelu je redukována na potřebné minimum
- **Spotřeba** – efektivní rozpoznávání, nízká velikost a nepotřebné připojení k síti vede ke snížení spotřeby energie

K efektivnímu redukování velikosti a zrychlení **inference** (rozpoznávání/klasifikace) modelu přispívá knihovna **FlatBuffer**.

Tento typ modelu jde natrénovat s pomocí nástroje **TensorFlow Lite Model Maker** nebo konvertovat klasický TensorFlow model nástrojem **TensorFlow Lite Converter**, kdy je výsledný model reprezentován koncovkou **tflite**. Vytvořený model pak můžeme využít nejen v operačních systémech Android a iOS, ale také v mikropočítačích založených na Linuxu.

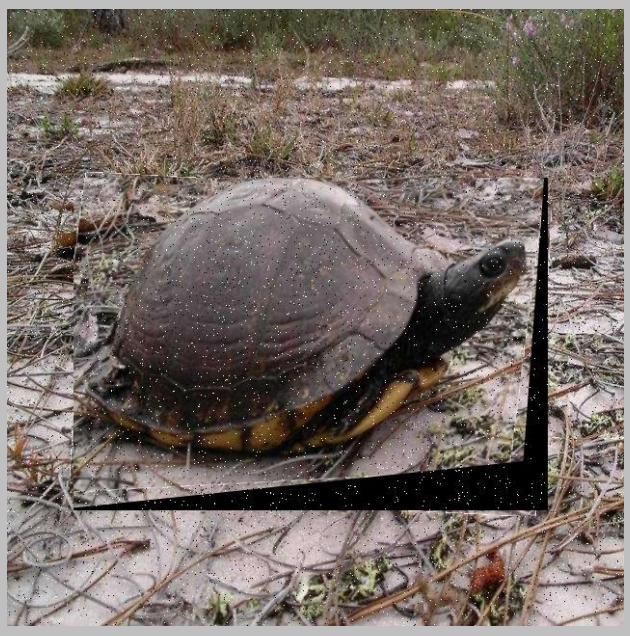
2.2.2 Problémy při učení

Tak jako i my lidé se při učení občas dostáváme do problémů a nepochopení látky, kterou se zrovna učíme, je tomu stejně tak i u učení CNN.

2.2.2.1 Trénovací data

Jedním z příkladů klasického problému při učení sítí může být nedostatek dat nebo jejich kvalita. Počet potřebných dat vždy závisí na robustnosti sítě a daném problému, kterému má síť porozumět a naučit se jej. Obecným pravidlem je alespoň 1000 obrázků do každé třídy, kterou se má síť naučit. [8]

Pro zajištění dostatečného počtu trénovacích dat lze využít technika **augmentace dat**, díky které můžeme trénovací data přetransformovat na nová data se změněnou podobou. Díky augmentaci vytvoříme nová data, která můžou mít jiný barevný kontrast, rotaci, měřítko anebo například nanesenou masku či šum.



Obrázek 4. Příklad augmentovaného obrázku.

2.2.2.2 Overfitting

Problém zvaný overfitting můžeme chápat jako **přeučení** sítě. Je charakteristické tím, že se síť příliš přizpůsobila trénovacím datům a nebude schopná určovat přesné výsledky klasifikace na nových datech, které ještě „neviděla“.

K monitorování tohoto jevu se při trénování využívá část datasetu zvaná **testovací** nebo **validační**. Jedná se o menší výseč z datasetu, která obsahuje data, která nesmí být využita při samotném trénování sítě, ale právě naopak se využívají pro validaci a posouzení správného učení. Během trénování se vždy po daných krocích (většinou jedné iteraci) tyto validační data aplikují na doposud naučenou síť a vrátí hodnocení kvality sítě v podobě metriky **loss**. Hodnota **loss** by měla být co **nejmenší** a udává chybovost, v jakou síť porozuměla validačním datům. [9]

Při tomto monitorováním můžeme jako jedno z možných řešení zamezení overfittingu využít tzv. **early stopping**. Předčasné ukončení trénování sítě provedeme v případě, když se hodnota loss udržuje v podobných hodnotách nebo začíná stoupat. V tomto momentě můžeme trénování zastavit a tím zamezit dalšímu učení, které by mohlo mít za následek špatnou generalizaci na nových datech. [9] Mějme však na paměti, že vynucení předčasného zastavení můžeme provést pouze tehdy, pokud při učení generujeme tzv. záchytné soubory obsahující stav doposud naučené sítě.

Dalším možným řešením je zajištění většího počtu trénovacích dat, například již zmíněnou augmentací, na kterých se síť může učit. To zapříčiní větší počet parametrů, které se musí síť naučit, a tudíž redukuje její možné přeučení.

Mezi často používanou techniku zaměřující se na zabránění přeučení spadá i **dropout** regulace. Jejím úkolem je ukončení spojení mezi náhodně vybranými aktivačními neurony v každé iteraci učení a zvyšuje tím robustnost celé sítě. [9] [10]

2.2.2.3 Underfitting

Underfitting je téměř opakem overfittingu. Jedná se o problém, který je specifický nedostatečným časem trénováním nebo příliš jednoduchou architekturou sítě, což opět způsobuje špatnou generalizaci na nových datech. [10]

Z tohoto lze chápat, že pokud se snažíme zabránit problému přeučení, můžeme se snadno dostat do opačného problému „nedoučení“ a zase naopak. Proto se při trénování sítě snažíme vždy nalézt „zlatou střední cestu“ pro zajištění kvalitních výsledků finálního modelu sítě.

2.2.3 True vs False a Positive vs Negative

Jedná se o 4 základní pojmy při klasifikování vstupních dat na naučeném modelu sítě využívající se pro hodnocení modelu.

Pro lepší pochopení následujících významů v tabulce (Tabulka 1.) berme v potaz, že je naše síť naučena rozpoznávat 2 druhy zvířat: slon a zebra.

Tabulka 1. Pojmy při klasifikování dat modelem.

Název pojmu	Vstupní obrázek	Klasifikace modelem	Vyhodnocení
True-Positive	Slon	Slon	SPRÁVNÉ
True-Negative	Jelen	-	SPRÁVNÉ
False-Positive	Nosorožec	Slon/Zebra	CHYBNÉ
False-Negative	Zebra	-	CHYBNÉ

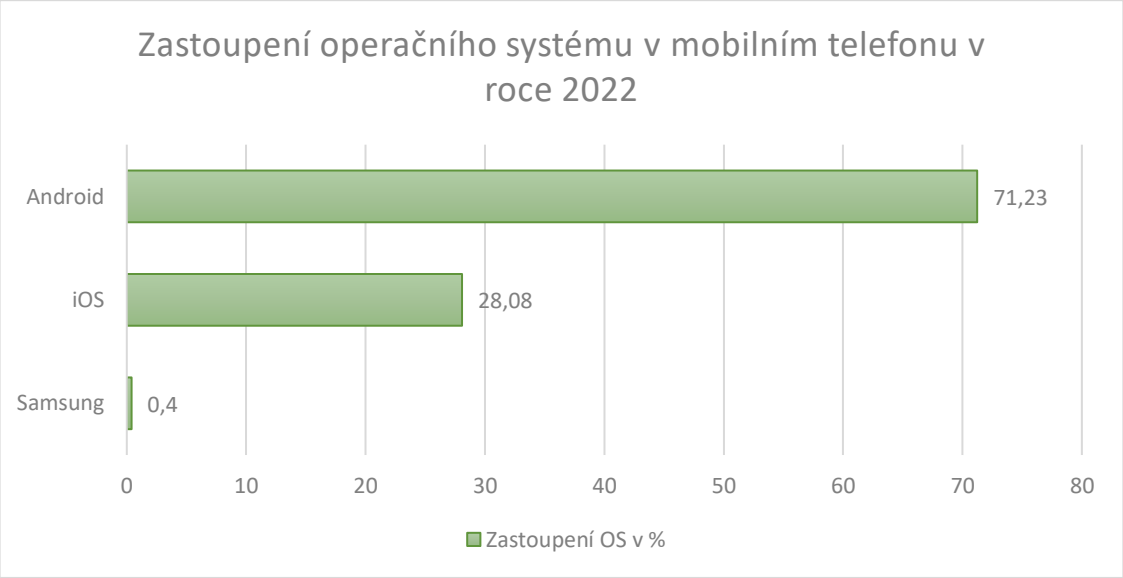
2.3 Druhy algoritmů

YOLO, SSD, EfficentNet/Det, …

3 VÝVOJ MOBILNÍCH APLIKACÍ

V dnešní době rozlišujeme 2 hlavní hráče na poli operačních systémů, pro které je možné vyvíjet mobilní aplikace. Jedná se o zařízení s operačním systémem iOS nebo Android. Z grafu (Graf 1.) lze vidět, že ve světě převládá operační systém Android a jelikož je tato práce zaměřena na vývoj mobilní aplikace na platformě Android, bude velká část této kapitoly věnovaná právě tomuto operačnímu systému.

Graf 1. Zastoupení mobilních operačních systémů za rok 2022. [11]



3.1 Druhy vývoje mobilních aplikací

3.1.1 Platformě závislé

3.1.1.1 Nativní vývoj

Nativní vývoj aplikací v praxi znamená, že vyvíjená aplikace bude spustitelná pouze na příslušné platformě operačního systému, pro který je aplikace naprogramovaná. Tento způsob vývoje převládá u takových aplikací, které se neobejdou bez veškerého výkonu a hardwaru mobilního telefonu, jako je například fotoaparát, čipy NFC nebo GPS, popřípadě přímá práce s paměťovým adresářem.

Při takovém vývoji je ale nutné vytvořit zvlášť aplikaci pro platformu Android a zvlášť aplikaci pro platformu iOS a díky tomu je celkový vývoj mnohdy časově i finančně náročnější.

3.1.2 Platformě nezávislé – multiplatformní

3.1.2.1 Webový vývoj

Přesným opakem nativního vývoje mobilních aplikací je webový vývoj. Jedná se o responzivní webovou stránku využívající webové technologie, jako je standard HTML 5 a JavaScriptové frameworky. Tyto aplikace většinou nemají přímý přístup k hardwaru mobilního telefonu, s výjimkou GPS nebo základní funkčnosti kamery. Jelikož takto vyrobené aplikace využívají pro svůj běh pouze internetový prohlížeč, není možné je samostatně nainstalovat do zařízení, ale podporují uložení na plochu zařízení.

3.1.2.2 Hybridní vývoj

Hybridní vývoj je dosti podobný k vývoji webovém. Opět se jedná o aplikaci využívající webové rozhraní internetového prohlížeče, které se nazývá WebView. Toto rozhraní je „zabaleno“ v nativní části aplikace, díky kterému je možné aplikaci nainstalovat na libovolné zařízení. K výše zmíněnému „zabalení“ se využívá tzv. wrapper technologie, například Ionic Capacitor, která se postará, aby webová aplikace dokázala komunikovat s hardwarem mobilního telefonu a tím využívat jeho funkce. [12]

3.2 Architektonické vzory

Slouží pro nastolení přístupu a pravidel při vývoji mobilní aplikace. Díky dodržení všech pravidel některého z vybraného vzoru, má aplikace pevně danou strukturu, ve které se dá snadno orientovat díky rozdělení do určitých celků a v budoucnosti rozšiřovat o novou implementaci.

Následující příklady vzorů spadají pod tzv. třívrstvou architekturu, která se již podle názvu vyznačuje třemi základními vrstvami:

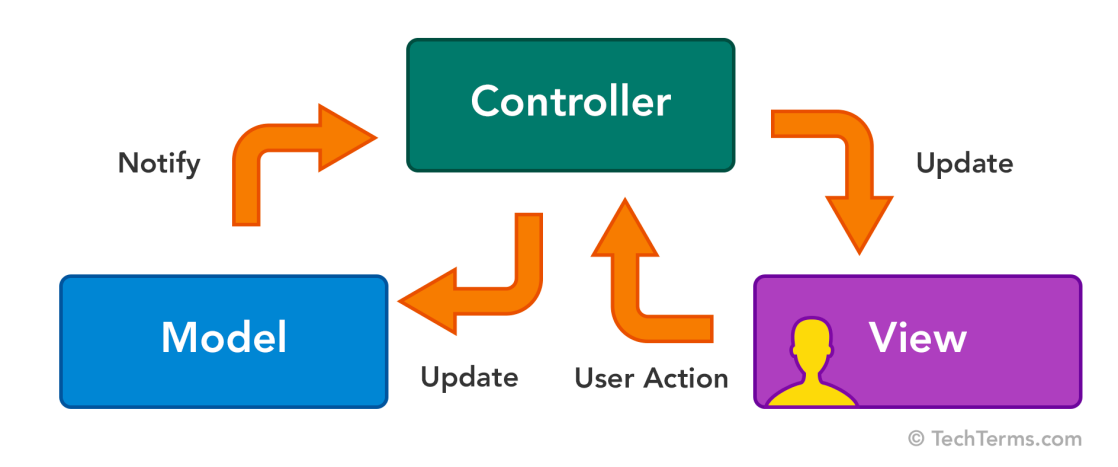
1. **Prezentační vrstva** se stará o prezentování/zobrazení uživatelského rozhraní uživateli dané aplikace a umožňuje mu tím aplikaci ovládat. Jedná se o platformě závislou vrstvu – webová, desktopová, mobilní.
2. **Aplikační vrstva** se stará o veškeré operace a výpočty v běhu aplikace. Zprostředkovává komunikaci mezi Datovou a Prezentační vrstvou. Zpracovává vstupy od uživatele, nebo naopak posílá data do Prezentační vrstvy pro zobrazení uživateli.

3. **Datová vrstva** je vrstva, která se stará o ukládání a práci s daty aplikace. Může se jednat o lokální nebo serverové úložiště nebo databázi. Tato data jsou vždy dostupná pro Aplikační vrstvu.

3.2.1 Model-View-Controller

Model-**V**iew-**C**ontroller neboli **MVC**, je jeden z nejčastějších architektonických vzorů při tvorbě webových aplikací. Jeho rozdělí do tří vrstev je následující: [13]

- Model – Tato část představuje logiku a práci s daty aplikace. Stará se o načítání, ukládání a zpracování dat z databáze a při tom nemá tušení o tom, jak jsou data na aplikaci závislá a zobrazená uživateli.
- View – View můžeme z angličtiny přeložit jako *pohled*, a to je přesně tím, čím je. Prakticky se jedná o grafické zobrazení aplikace s daty, které View čerpá z Modelu. Tato vrstva by nikdy neměla být schopná měnit data aplikace a vykonávat jinou logiku, než je zobrazování šablony s UI.
- Controller – Kontrolér většinou reaguje na události uživatele nebo samotného systému. Komunikuje s Modelem a na základě vstupu od uživatele rozhoduje, které View aplikace se načte, popř. s jakými daty.



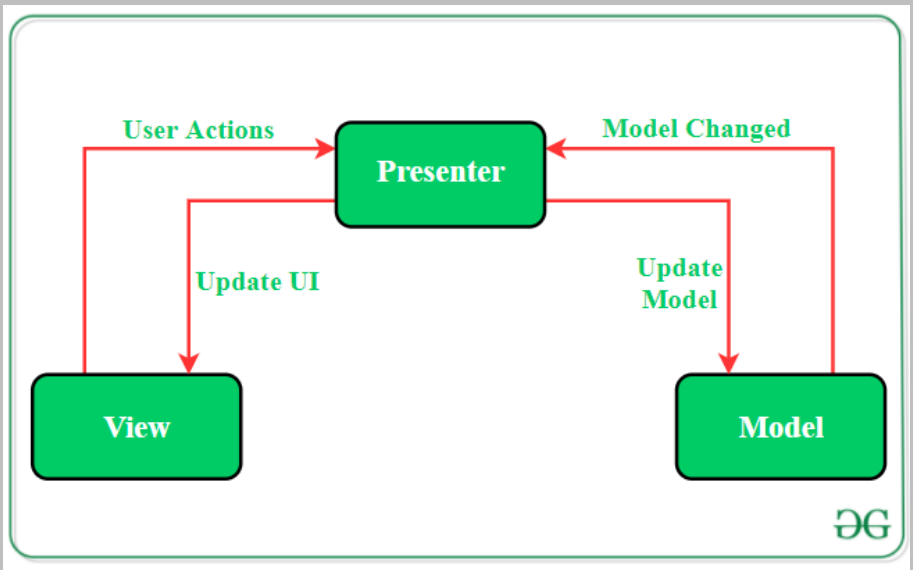
Obrázek 5. MVC diagram. [14]

Z tohoto rozdělení je patrné, že všechny vrstvy jsou na sobě nezávislé a rozdělené na bloky, které souvisí pouze spolu. Díky tomu je aplikaci možné snadno škálovat a modifikovat bez obav porušení některé z existující funkcionality aplikace.

3.2.2 Model-View-Presenter

Architektonický vzor MVP vychází ze vzoru MVC a díky tomu mají téměř identické chování. Model opět reprezentuje datovou vrstvu, View zobrazuje data uživateli a také reaguje na požadavky uživatele, které posílá na svůj Presenter, který se navíc stará o komunikaci mezi vrstvami Model a View.

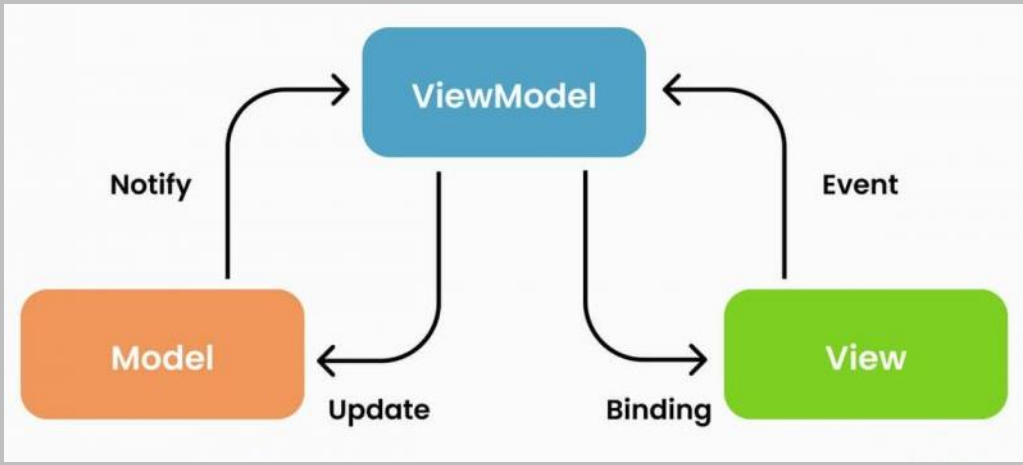
Změna od MVC přichází u samotné komunikaci mezi View a Modelem, které oproti MVC nejsou nyní napřímo propojeny, a tudíž musí Presenter předat data z Modelu do View sám a tím samotný View naformátovat. To umožňuje větší flexibilitu při automatizovaném testování. [15]



Obrázek 6. MVP diagram. [15]

3.2.3 Model-View-ViewModel

Model-**V**iew-**V**iew**M**odel (MVVM) je silným architektonickým vzorem pro tvorbu rozsáhlých, nejen, mobilních aplikací. Spočívá ve vytvoření vazeb mezi daty a uživatelským rozhraním. Tato vazba probíhá za pomocí binding třídy ve ViewModelu a stará se o držení dat, jejich aktualizování a přeposílání na View. To znamená, že z UI je zcela odstraněn kód aplikace a data jsou pevně vázána právě na třídu ve ViewModelu, kde si neustále uchovává aktuální stav.



Obrázek 7. MVVM diagram. [16]

- **Model** představuje identickou funkci jako u vzorů MVC a MVP, to je, že se stará o získávání dat z databáze aplikace.
- **View** popisuje uživatelské rozhraní, stará se o komponenty, které jsou ovlivněné vstupy od uživatele. Nemělo by obsahovat aplikační logiku.
- **ViewModel** – nejdůležitější část tohoto arch. vzoru. Obsahuje veškerou aplikační logiku, stará se o udržování stavu a aktualizování dat podle potřeb View.

Jelikož ViewModel nemusí být pevně vázaný na konkrétní View, může být tak automaticky použito ve více případech a tím značně redukovat celý kód aplikace, kterou je pak možné snadno testovat a rozšiřovat. [17]

3.3 Android

Jedná se o open-source projekt založený na Linuxovém jádru vyvíjeným společností Google od roku 2008. [18] Dá se považovat za krále mobilních operačních systémů, jelikož se jedná o nejvíce používaný operační systém v mobilních zařízení a před druhým nejčastějším operačním systémem iOS si každoročně drží odstup přibližně 43 % využívaných zařízení. [11]

Aplikace vyvíjené pro tento systém se programují v jazyku Java nebo v moderním jazyce Kotlin. Vývojářům jsou dostupné různé nástroje v podobě SDK balíčků (neplést s API), které hrají důležitou roli při vývoji jakékoliv aplikace. Tyto SDK nástroje jsou běžnou součástí oficiálního vývojového prostředí Android Studio.

TODO (api levely, třeba Android Studio)

3.4 Framework Jetpack Compose

Jedná se o novou moderní technologii pro tvorbu nativních mobilních aplikací pro platformu Android s pomocí programovacího jazyka Kotlin. Jeho hlavní předností je rychlejší vykreslování prvků, intuitivní a rychlejší tvorba UI elementů aplikace a tím i plynulejší chod celé aplikace.

TODO

II. PRAKTICKÁ ČÁST

TODO, základní úvod a seznámení s tím, jak se aplikace začala vyvíjet, co čeká čtenáře v dalších kapitolách

4 TVORBA DATASETU

Dataset pro CNN se dá chápat jako obrovská sada obrázkových dat, na kterých se trénuje neuronová síť. Tyto obrázky se musí setřídit do společných tříd (složek), které spolu souvisí a nevznikala tak pouze nepojmenovaná hromada dat, pro kterou by nebylo využití. Z tohoto jasně vyplývá, že se jedná o poměrně těžkou manuální práci, která vyžaduje píli, přesnost a preciznost, aby nevznikaly chyby, které by zapříčinily nesprávnému trénování sítě.

K natrénování aktuálních modelů byly vytvořeny prakticky 2 datasety.

- První z nich je zaměřený na již zmíněný druh strojového učení s využitím CNN pro **Image Classification**. Tento typ datasetu vyžaduje roztříděné obrázky na třídy, které se v nich vyskytují. Tyto třídy jsou určeny pojmenovanou složkou, do nichž se dané obrázky umístí a tím vznikne následující hierarchie:
  - Elephant
    - obrázek1.jpg, obrázek2.jpg, …
  - Zebra
    - obrázek1.jpg, obrázek2.jpg, …
  - …
- Druhý typ datasetu je pro CNN určenou na **Object Detection**. Tento druh datasetu je typický tím, že ke každému obrázku se musí vyskytovat speciální soubor, který obsahuje souřadnice hledaných objektů z daného obrázku. Díky tomu je celý proces tvorby tohoto datasetu mnohem obtížnější a časově náročnější než u prvního.

Dataset tvoří jádro celé práce, jelikož bez něj by se nedala natrénovat neuronová síť, která by byla schopná zvířata rozpoznat. Pro jeho přípravu bylo využito několik technik a nástrojů, které pomohli velkou část jeho tvorby urychlit automatizací.

4.1 Image Classification dataset

V této sekci se zaměříme především na vznik a vývoj datasetu pro typ CNN zaměřené na klasifikaci obrazu. Jak již víme z textu výše (TODO, nadpis), jedná se o klasifikaci obrázku, který síť klasifikuje jako jeho celek do třídy, kterou je naučena a schopna identifikovat.

Tento druh datasetu vyžaduje poměrně velký počet trénovacích dat, které musí být roztříděné na třídy v přibližně rovnoměrném rozdělení, k čemu dopomohou různé nástroje.

4.1.1 Použité techniky

Využitých nástrojů a technik při tvorbě datasetu pro image classification bylo několik. Důležitým prostředníkem hráli naprogramované skripty v jazyce Python a aplikace *Flickr* a jeho API, se kterou se seznámíme níže. Díky tomu bylo alespoň částečně možné automatizovat složitý a časově vyčerpávající proces, který by manuálně snad nešel ani dokončit.

4.1.1.1 Flickr a jeho API

Hlavním zdrojem obrázků pro tuto práci byla webová aplikace https://www.flickr.com/, která obsahuje miliony obrázků označeny tagy, které dopomáhají jejich přesnému vyhledání a možnosti stažení. Ruční stahování tisíce obrázků je poměrně nemyslitelná záležitost, na kterou v aplikaci mysleli a pro vývojáře vytvořili šikovnou API. K jejímu získání se stačí na stránce zaregistrovat a nechat si zdarma vygenerovat její klíč, který je potřebný ve scriptu sloužící právě pro stahování obrázků.

Script byl napsaný v programovacím jazyce Python 3.7.9 a stará se o stahování obrázků zvířat, které si programátor zvolí. V první verzi script pouze stahoval obrázky a nic jiného nerešil, což při následné ruční filtraci dělalo potíže, jelikož bylo zjištěno, že některé obrázky jsou staženy například 8krát, protože je tam lidé několikrát nahráli. Proto byl script upraven tak, aby po stažení obrázku zkontroloval jeho binární hash s ostatními staženými obrázky a pokud by se již daný hash někde vyskytoval, právě stažený obrázek vymaže. Navíc se díky tomu redukoval celkový počet originálních obrázků, které bylo možné stáhnout a hrozilo v zacyklení „stažení – vymazání“ a proto byla přidána další kontrola pro opakované vymazání staženého obrázku, kdy se po 60 cyklech stahování přeruší a popřípadě se přejde na další druh zvířete.

Stažené obrázky se museli ještě ručně vyfiltrovat, jelikož se v sadě vyskytovali takové, které nemají s daným zvířetem nic společného (například automobil značky Jaguar), nebo se v obrázku nacházelo v minimálním měřítku. Po vyfiltrováním bylo u každého druhu zvířete kolem 2000–3000 obrázků, což bylo pro klasifikaci poměrně málo.

4.1.1.2 Image augmentation

Pro vyřešení problému s nízkým počtem obrázků existuje technika nazvaná augmentace dat. Díky ní je možné z malé sady obrázků vytvořit sadu větší, a to s využitím transformací originálních obrázků.

Využité transformace:

- Zrcadlení obrázku horizontálně.
- Zrcadlení obrázku vertikálně.
- Posun obrázku nahoru nebo dolů.
- Otočení obrázku v rozmezí -30° až 30°.
- Změna měřítka obrázku v rozmezí 75 % až 130 % originální velikosti.

S použitím této techniky bylo vygenerováno přibližně 6000 obrázků do každé kategorie pro zajištění kolem 8000 obrázků každého zvířete, což už je slušný počet pro trénování CNN zaměřené na klasifikaci.

4.1.2 Finální zhodnocení a informace

Dataset se neustále měnil a přetvářel, přibývaly jiné druhy zvířat, popřípadě byly odstraněny ty, které neobsahovaly správná data pro dostatečně kvalitní trénování modelu. Tato fáze tvoření probíhala zcela paralelně se samotnou tvorbou modelu a jeho učením, kdy bylo potřeba dolaďovat různé detaily datasetu pro zajištění co největší přesnosti naučeného modelu.

Finální podoba datasetu obsahuje TODO druhů zvířat s celkovým počtem X obrázků přesahující X GB dat.

Tady třeba ještě vložit tabulku, ve které bude druh zvířete a počet obrázků.

4.2 Object Detection dataset

Tato sekce obsahuje detailní popis tvorby datasetu využívající CNN s technikou object detection. Vytvoření tohoto typu datasetu vyžaduje mnohem více úsilí a času než u předchozího typu. Je to díky tomu, že aby se CNN mohla lépe učit nalézt objekt v obraze, musíme jí tento objekt v trénovacích datech vyznačit, což již značí spoustu ruční práce, kterou nelze jednoduše automatizovat.

4.2.1 Použité nástroje

I při tvorbě tohoto druhu datasetu bylo využito několik nástrojů, převážně v podobě Python skriptů, které práci mnohonásobně urychlili nebo pomohli opravit chyby v desítkách tisíců souborů. Veškeré vytvořené anotace jsou ve speciálním formátu zvaném VOC Pascal, který

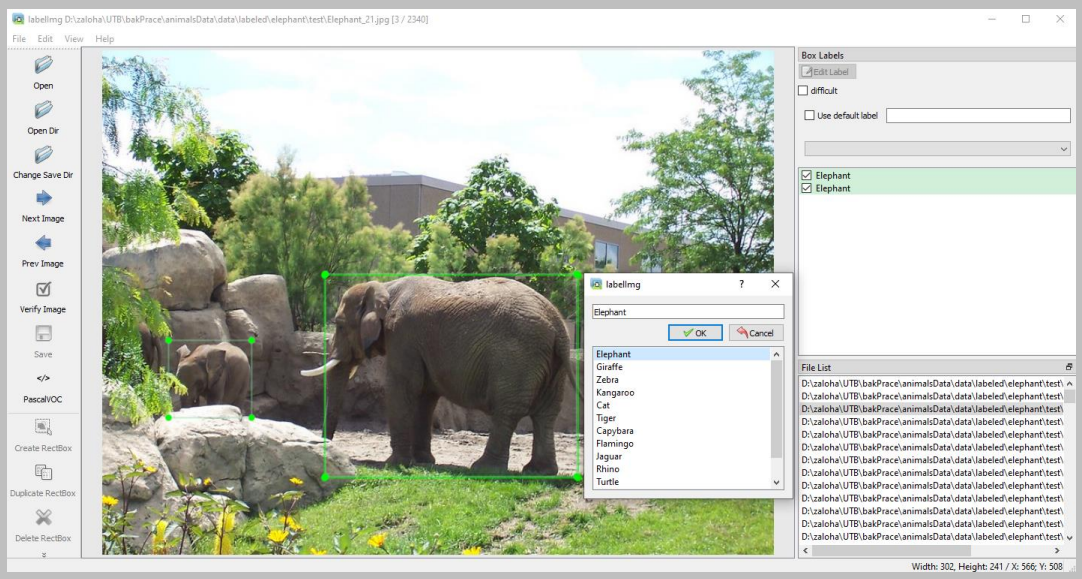
je potřebný knihovnou TensorFlow použitou pro trénování. Data tohoto formátu jsou uložena v souborech s koncovkou XML v následujícím formátu:

```
<?xml version="1.0" encoding="UTF-8"?>
- <annotation>
    <filename>Elephant_10.jpg</filename>
  - <size>
      <width>800</width>
      <height>532</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>Elephant</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>161</xmin>
        <ymin>4</ymin>
        <xmax>800</xmax>
        <ymax>532</ymax>
      </bndbox>
    </object>
</annotation>
```

Obrázek 8. Anotace ve formátu VOC Pascal XML.

4.2.1.1 Program LabelImg

Prvním důležitým nástrojem je program LabelImg. Jedná se o grafický nástroj pro anotaci a označování objektů v obrázku. Je napsán v programovacím jazyce Python a pro všechny zcela zdarma dostupný pod instalačním příkazem *pip install labelImg*.



Obrázek 9. Ukázka prostředí programu LabelImg.

V tomto programu bylo ručně vytvořeno pár tisíc anotací obrázků několika málo druhů zvířat, jejichž obrázky byly převzaty z již dříve staženého datasetu pro image classification.

Tato část tvorby si vyžádala nejvíce času, a i přes to nebylo možné bez získání většího počtu dat pokračovat, a proto přišlo na řadu stahování volně dostupných datasetů se zvířaty.

4.2.1.2 Volně dostupné datasety

Pro zajištění dostatečně velkého počtu obrázků bylo zapotřebí najít a stáhnout volně dostupná řešení jiných vývojářů. Mezi známé portály obsahující datasety se řadí především stránka https://www.kaggle.com/ a https://roboflow.com/. Na nich je možné zdarma stáhnout vytvořené datasety od jiných uživatelů a použít je tak ve vlastním zájmu.

V této práci to nebylo výjimkou a několik datasetů bylo staženo: dát do přílohy

- Kapybara
    - https://github.com/freds0/capybara_dataset
    - https://universe.roboflow.com/miguel-narbot-usp-br/capybara-and-animals/dataset/1
- Klokan
    - https://www.kaggle.com/datasets/hugozanini1/kangaroodataset?resource=download
    - https://github.com/experiencor/kangaroo
    - https://universe.roboflow.com/z-jeans-pig/kangaroo-epscj/dataset/1
- .....

Dohromady bylo staženo kolem 50_000 obrázků, které bylo potřeba vyfiltrovat, jelikož spoustu z nich bylo vadných, a nakonec je nebylo možné použít. Takovéto obrázky byly většinou příliš zmenšené, obsahovaly chybné nebo dokonce žádné označení zvířat anebo byly záběry pořízené z fotopastí, ve kterých bylo zvíře zachycené ze špatných úhlů.

TODO, jednotlivé skripty asi s odkazem do příloh

4.2.1.3 Skript pro kontrolu vadných obrázků

V této práci byl využitý Python skript, který zkontroloval všechny obrázky v datasetu, pro zajištění jejich správného formátu JPEG. Tento skript byl použit poté, co se během trénování objevila chyba označující, že dataset obsahuje poškozený obrázek. Bohužel ale tato chyba

neobsahovala konkrétní obrázek a vyhledat jej ručně je prakticky nemožné. Skript je výtvorem Yasooba Khalida a lze jej najít například na tomto odkazu: https://stackoverflow.com/a/65367773.

Skript byl lehce upraven, aby místo odstranění vadného obrázku vypsal pouze jeho jméno a následně byl obrázek s jeho příslušným XML souborem smazán ručně. Díky tomuto skriptu bylo nalezeno celkově asi 12 poškozených obrázků, které během trénování modelu zapříčiňovali jeho pád.

4.2.1.4 Skript pro přejmenování souborů

Během úprav datasetu bylo vždy několik obrázků staženo nebo odstraněno, což značilo nekonzistentnost v názvů souborů. Také názvy souborů ze stažených datasetů obsahovaly pouze směsici písmen a čísel, na které nebyla radost pohledět. Z tohoto důvodu by vytvořený skript, který přejmenoval všechny soubory podle představ autora, a to ve tvaru pro vlastní obrázky: *zvire_cislo.jpg* a ve tvaru pro stažené obrázky: *zvire_download_cislo.jpg*. Tento skript dále musí brát v potaz taky XML soubory příslušných obrázků, jejichž jména musí být až na koncovku totožná.

4.2.1.5 Skript pro editaci XML souborů

Tento skript byl vytvořen za účelem aktualizování tagu *filename* po aplikování skriptu se změnou názvů souborů. Postupem času byl aktualizován na vyhledávání chybných anotací ze stažených datasetů, jelikož mnohé z nich obsahovali anotované objekty o velikosti menší než 33 pixelů na šířku nebo výšku, což je pro TensorFlow hraniční hodnota minimální velikosti objektu v obrázku. Pokud se počet objektů v XML souboru rovnal 0, byl tento soubor spolu s příslušným obrázkem přesunut do složky, která obsahuje všechny data ve vadném formátu pro TensorFlow.

Dále se ve stažených datasetech objevoval problém v tom, že byl objekt anotován o 1 pixel až za hranice samotného obrázku, tudíž byl skript doplněný o opravu i těchto hranic.

Skript taky odstraňuje nepotřebné tagy *path, folder, source, occuled, polygon* a tag *pose* nastavuje na *Unspecified*.

Pokud soubor obsahuje anotovaný objekt, který uživatel nepotřebuje, lze jej tímto skriptem odstranit.

4.2.1.6 Skript pro odstranění malých obrázků

Mezi staženými datasety se často objevovali obrázky, jejichž velikost byla příliš malá. Tento skript se stará o přesunutí takových obrázků do již zmíněné složky, která obsahuje vadná data. Prahová hodnota obrázků je nastavena na 331 pixelů do šířky i výšky a vše co je menší bylo přesunuto.

4.2.1.7 Skript pro dělení datasetu

4.2.1.8 Skript pro tvorbu TFRecords

TODO, opět nějaké zhodnocení jako u Image Classification datasetu

5 TRÉNOVÁNÍ MODELU

Tato část popisuje celkový vývoj modelu, který je v aplikaci využitý pro rozpoznávání zvířat.

5.1 Test dostupných modelů

5.2 API pro vytvoření vlastních modelů

5.2.1 Model pro klasifikaci

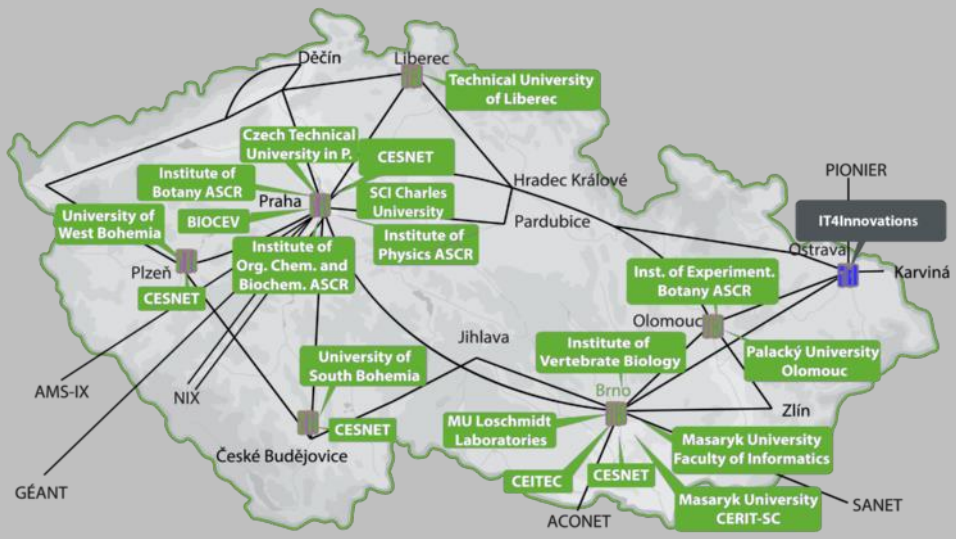
5.2.1.1 TensorFlow lite model maker

5.2.2 Model pro detekci

5.2.2.1 TensorFlow 2 Object Detection API

5.3 MetaCentrum

MetaCentrum (**MetaVO**) je virtuální organizace České republiky zabývající se poskytováním vysoce výkonného hardwaru ve formě Gridových výpočtů[2], studentům a akademickým pracovníkům ve členském sdružení CESNET zdarma. [19]



Obrázek 10. Mapa propojených míst spadajících do MetaCentrum. [19]

Pro finální trénování použitého modelu ve vytvořené aplikaci byla použita právě tato služba, která výrazně dopomohla snížení času trénování, a hlavně vytíženosti osobního počítače autora této práce.

5.3.1 Seznámení a prvotní nastavení

Samotná registrace probíhá klasicky jako na každé jiné stránce, ale s využitím školního emailu studenta, který je spravován federací **eduID.cz** spadající pod již zmíněné sdružení CESNET.

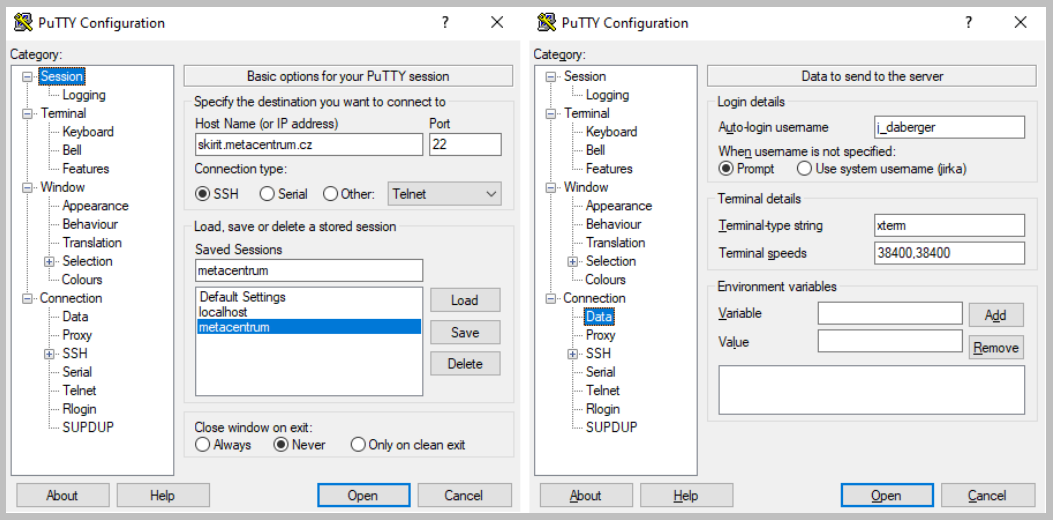
K plnému využití grafických karet při trénování modelu jsou zapotřebí knihovny CUDA, které jsou dostupné na vybraných grafických kartách společnosti NVIDIA. Abychom je ovšem mohli využít, je potřeba zaregistrovat se jako developer na stránkách NVIDIA (https://developer.nvidia.com/) a souhlasit s jejich licencí. V MetaVO jsou pro nás tyto a mnohé další licence připraveny a stačí s nimi souhlasit.

[2] Gridové výpočty znamenají propojení několika PC z různých lokací pro dosažení co největšího výpočetního výkonu.

5.3.1.1 *Konfigurace nástroje PuTTy*

Po registraci a přihlášení již můžete dostupné výpočetní zdroje využívat pro své potřeby. Pro přístup na samotná zařízení se dá na počítačích s operačním systémem Windows použít volně dostupná aplikace **PuTTy**, která přes okno terminálu obstará připojení na vzdálenou síť MetaVO, která využívá operační systém Linux.

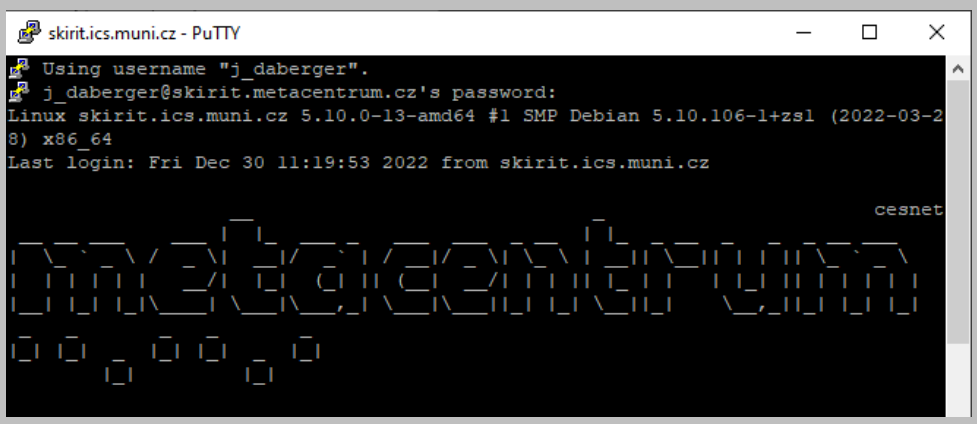
Pro pohodlné a rychlé přihlášení lze uložit nastavení přihlašovací konfigurace, abychom vše nemuseli dělat manuálně vždy, když se chceme na síť MetaVO přihlásit.



Obrázek 11. Nastavení přihlášení v PuTTy.

Na obrázku (Obrázek 11.) lze na levé straně v sekci „Session“ vidět, že celá konfigurace je uložená pod názvem „metacentum“. Adresa, na kterou se uživatel připojí je „skirit.metacentrum.cz“ s portem 22 a typ připojení je zabezpečený kanál SSH. V sekci „Connection/Data“ lze taky nastavit login uživatele, který se chce přihlásit.

Díky tomuto nastavení se uživatel může kdykoliv rychle přihlásit bez nutnosti zadávat vše znovu. Nakonec jen zadá své heslo a přihlášení do sítě je hotové.



Obrázek 12. Úspěšné přihlášení do sítě programem PuTTy.

Toto terminálové okno je určeno jako hlavní a taky jediná možnost pro práci s výpočetními stroji. Uživatel je zprvu přihlášen do tzv. **čelního uzlu**, který slouží jako jakýsi rozcestník a k základní manipulaci v uživatelském úložišti, a proto je na něm zakázáno provádět výkonově náročné operace.

Takové operace se již provádějí na výpočetních strojích k tomu určených. Pro připojení na stoje je potřeba vytvoření úlohy z čelního uzlu.

- **Interaktivní** úloha slouží pro samotné přichystání prostředí, ve kterém bude uživatel dále pracovat. Touto úlohou se uživateli přidělí požadovaný stroj a uživatel může manuálně provádět operace potřebné k nachystání svého prostředí.
- **Dávková** úloha se stará o přímé zpracování konkrétního scriptu, který si uživatel vytvoří předem. Po přidělení požadovaného stroje se začne úloha provádět sama bez dalšího vstupu od uživatele.

Příklady použití obou úloh nalezneme níže v textu.

5.3.1.2 Instalace potřebných knihoven

První kroky v MetaVO byly poměrně chaotické, a i přes poměrně rozsáhlou dokumentaci ve formě wiki stránek (https://wiki.metacentrum.cz/wiki/Hlavn%C3%AD_strana) se nedařilo knihovny TensorFlow zprovoznit. Z těchto důvodů byla kontaktována uživatelská podpora, která velice ochotně poradila a pomohla zprovoznit základní instalaci.

Pro tuto instalaci již byla potřebná **interaktivní** úloha, která byla spuštěna následujícím příkazem:

qsub -l walltime=4:0:0 -q default@meta-pbs.metacentrum.cz -l select=1:ncpus=1:mem=8gb:scratch_local=10gb -I

Tímto příkazem se požaduje stroj s jedním CPU, 8 GB RAM a 10 GB lokálního úložiště, který bude uživateli dostupný po dobu 4 hodin.

```
MISTO_INSTALACE=/storage/brno2/home/j_daberger/obj_detect
cd $MISTO_INSTALACE
export TMPDIR=$SCRATCHDIR
curl -Ls https://micro.mamba.pm/api/micromamba/linux-64/latest | tar -xvj bin/micromamba
eval "$(./bin/micromamba shell hook -s bash)"
MAMBA_EXE=bin/micromamba micromamba create -p $(pwd)/tf2 -c conda-forge python==3.8 pycocotools libprotobuf==3.19.4 cudatoolkit=11.2 cudnn=8.1.0
MAMBA_EXE=bin/micromamba micromamba activate $(pwd)/tf2
git clone https://github.com/tensorflow/models TensorFlow/models
cd TensorFlow/models/research/
cp object_detection/packages/tf2/setup.py .
pip install .
protoc object_detection/protos/*.proto --python_out=$(pip show pycocotools|awk '/^Location:/{print $2;}')
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$CONDA_PREFIX/lib/
export XLA_FLAGS=--xla_gpu_cuda_data_dir=$MISTO_INSTALACE/tf2/lib
export TF_XLA_FLAGS="--tf_xla_auto_jit=2"
clean_scratch
```

Výše uvedené linuxové příkazy mají za úkol vytvořit virtuální prostředí **micromamba** nazvané „tf2“ v domovském adresáři uživatele. Virtuální prostředí umožňuje instalaci různých balíků a knihoven, aniž by „udělalo nepořádek“ v normálním prostředí uživatele, což by mohlo mít za následek nekompatibilitu některých knihoven a tím jejich nefunkčnost. Do vytvořeného prostředí byly dále nainstalovány potřebné knihovny pro práci s frameworkem **TensorFlow 2 Object Detection API** i s podporou GPU pro rychlejší trénování. Z oficiálního GitHubu byl naklonován a nainstalován repozitář tohoto frameworku. Pro finální zprovoznění je zapotřebí vytvořit speciální **proměnné** prostředí, které obsahují cesty k důležitým souborům, které jsou frameworkem požadovány pro jeho fungování.

Toto, v konečném výsledku poměrně jednoduché, nastavování nakonec zabralo téměř 3 dny aktivního zkoumání, jelikož se objevovala celá řada problémů, z níž největší byla právě proměnná s názvem ***XLA_FLAGS***.

6 REALIZACE

7 TESTOVÁNÍ

7.1 Srovnání s dostupným řešením

ZÁVĚR

Text

SEZNAM POUŽITÉ LITERATURY

[1] Co je OCR. In: *Nanonets* [online]. [cit. 2023-01-12]. Dostupné z: https://nanonets.com/blog/what-is-optical-character-recognition

[2] COUMAR, Nanda. OCR. In: *Medium* [online]. [cit. 2023-01-12]. Dostupné z: https://medium.com/@nandacoumar/optical-character-recognition-ocr-image-opencv-pytesseract-and-easyocr-62603ca4357

[3] *Pokročilé techniky neuronových sítí.* [online]. In: . [cit. 2022-12-20]. Dostupné z: https://course.elementsofai.com/cs/5/3

[4] YAMASHITA, R., M. NISHIO a R.K.G. DO. Konvoluční neuronové sítě. In: *Convolutional neural networks: an overview and application in radiology* [online]. 2018, s. 611-629 [cit. 2022-12-20]. Dostupné z: doi:https://doi.org/10.1007/s13244-018-0639-9

[5] MU, Li, Zachary LIPTON, Zhang ASTON a Alexander SMOLA. Dopředná a zpětná propagace. In: *Dive into Deep Learning* [online]. 2021, s. 224-227 [cit. 2023-01-13]. Dostupné z: doi:https://arxiv.org/abs/2106.11342v3

[6] JOHNSON, Daniel. Informace o TensorFlow knihovně. In: *GURU99* [online]. 2022 [cit. 2023-01-15]. Dostupné z: https://www.guru99.com/what-is-tensorflow.html

[7] TensorFlow graph. In: *TensorFlow* [online]. [cit. 2023-01-15]. Dostupné z: https://www.tensorflow.org/guide/intro_to_graphs

[8] DERNONCOURT, Franc. Počet trénovacích dat. In: *StackExchange* [online]. 2016 [cit. 2023-01-17]. Dostupné z: https://stats.stackexchange.com/questions/226672/how-few-training-examples-is-too-few-when-training-a-neural-network/226693#226693

[9] Přeučení sítí. In: *IBM* [online]. [cit. 2023-01-17]. Dostupné z: https://www.ibm.com/topics/overfitting

[10] RUIZENDAAL, Rutger. Problémy s učením sítě. *Towards Data Science* [online]. [cit. 2023-01-17]. Dostupné z: https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d

[11] Zastoupení mobilních operačních systémů. In: *Statcounter* [online]. [cit. 2022-12-18]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202111-202211-bar

[12] VALA, Radek. *Programování mobilních aplikací: Metody vývoje mobilních aplikací* [Prezentace]. Fakulta aplikované informatiky - Univerzita Tomáše Bati ve Zlíně, 2022.

[13] POP, Dragos-Paul a Adam ALTAR. Vzor MVC. In: *Designing an MVC Model for Rapid Web Application Development* [online]. 2014, s. 1172-1179 [cit. 2022-12-18]. ISSN 1877-7058. Dostupné z: doi:https://doi.org/10.1016/j.proeng.2014.03.106

[14] MVC diagram. In: *TechTerms* [online]. [cit. 2022-12-18]. Dostupné z: https://techterms.com/definition/mvc

[15] MISHRA, Rishu. Difference Between MVC and MVP Architecture Pattern in Android. In: *GeeksForGeeks* [online]. 2020 [cit. 2022-12-18]. Dostupné z: https://www.geeksforgeeks.org/difference-between-mvc-and-mvp-architecture-pattern-in-android/?ref=gcs

[16] MVVM diagram. In: *Bach Khoa-npower* [online]. [cit. 2022-12-19]. Dostupné z: http://bachkhoa-npower.vn/mvvm-la-gi/

[17] MISHRA, Rishu. MVVM (Model View ViewModel) Architecture Pattern in Android. In: *GeeksForGeeks* [online]. 2022 [cit. 2022-12-19]. Dostupné z: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/

[18] Android. In: *Wikipedia* [online]. [cit. 2022-12-10]. Dostupné z: https://cs.wikipedia.org/wiki/Android_(opera%C4%8Dn%C3%AD_syst%C3%A9m)

[19] *MetaCentrum* [online]. [cit. 2022-12-30]. Dostupné z: https://metavo.metacentrum.cz/cs/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SDK Software Development Kit

IDE Integrated Development Environment

API Application Programming Interface

UI User Interface

OS Operating System

iOS iPhone Operating System

NFC Near Field Communication

GPS Global Positioning System

SMS Short Message Service

HTML Hypertext Markup Language

Tzv tak zvaný

Popř popřípadě

TODO

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: NÁZEV PŘÍLOHY