Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra Informatiky a kvantitativních metod

Rozpoznávání objektů a rozšířená realita

Diplomová práce

Autor: Roman Vlček
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové srpen 2018

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 30.7.2018 Roman Vlček

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Pavlu Křížovi, Ph.D. za metodické vedení práce a poskytnutí potřebných informací k jejímu vypracování.

Anotace

Práce se zabývá možnostmi využití frameworku pro rozšířenou realitu v mobilní hře určené pro sběr rádiových fingerprintů. Cílem je provést rešerši existujících řešení pro rozpoznávání objektů a rozšířenou realitu, a následný návrh použití jednoho z frameworků v dané mobilní hře. Výsledkem této práce by měla být mobilní aplikace, rozpoznávající jednotlivé štítky nacházející se u učeben na budově FIM. Dále by aplikace měla mít funkci pro určení místa, na kterém se štítek v nasnímaném obraze reálně nachází.

Práce je rozdělena do několika částí. V první části bude čtenář seznámen s již existujícími knihovnami pro rozšířenou realitu a rozpoznávání objektů, s jejich možnostmi, výhodami a nevýhodami. V druhé části bude provedena analýza dostupných řešení a samotného využití konkrétního frameworku pro danou mobilní hru. Poté bude následovat část týkající se návrhu a implementace frameworku do mobilní aplikace. V závěru práce bude část věnovaná testování spolehlivosti rozpoznávání objektů a zhodnocení výsledků zvoleného řešení.

Annotation

Title: Object Recognition and Augmented Reality

This thesis deals with possibilities of using augmented reality framework in mobile game designed to collect radio fingerprints. The aim is to research existing solutions and frameworks for augmented reality and object recognition. Second aim is to create mobile application that will use one of these frameworks. The result of this thesis should be a mobile application that can recognize the individual labels at the doors on FIM building. Next feature of this application will be identification of label location in the captured image.

This thesis is divided into several parts. In the first part, the reader will be introduced to augmented reality and image recognition frameworks. There will be descriped their features, advantages and disadvantages. In the second part, there will be an analysis of described solutions and usage of choosen framework in an Android application. Third section is about implementation chosen solution in an Android application. Next section will be about testing object recognition reliability

of implemented solution. At the end there will be an evaluation of results and recommendations for next sequel.

Obsah

Seznam obrázků

Seznam tabulek

Seznam pojmů

Bottleneck – set hodnot, pomocí kterých klasifikátor rozeznává obrázky od sebe

CPU – procesorová jednotka počítače

CUDA – technologie pro spouštění programů v jazyce C/C++ na grafickém procesoru od společnosti NVidia

Framework – sada programových kódů poskytujících nějakou funkcionalitu z konkrétní oblasti

GPS – družicový systém určený k navigování a určování pozice na Zemi

GPU – grafický procesor (výpočetní část grafické karty)

Image detection - metoda pro určení objektu v nasnímaném obraze - často též image recognition nebo klasifikace obrazu

ImageNet – databáze obrázků používaná k testování nových modelů

Inference Graph – graf pro odvozování, vytvořený z nějakého checkpointu při trénování modelu object detection při použití frameworku TensorFlow

Object detection – metoda (funkce) frameworku TensorFlow sloužící k určení umístění objektu ve snímaném obraze

OCR – technologie pro převod obrazu na text

OpenGL – standart specifikující rozhraní pro tvorbu počítačové grafiky

PASCAL VOC – formát XML dokumentu navržený pro obsažení dat potřebných k detekci objektů

Python – skriptovací jazyk, ve kterém je napsána většina skriptů používaných při práci s frameworkem TensorFlow

Rádiový fingerprint – bezdrátové sítě dostupné z uživatelova zařízení, využívá se k lokalizaci uživatele pomocí známé polohy WiFi sítí

Rozšířená realita (AR) – promítnutí virtuálních objektů do reálného světa

SDK – sada vývojových nástrojů umožňující tvorbu aplikací pro nějakou platformu

SLAM – funkce frameworku EasyAR pro sledování pohybu kamery

Tensor – vícerozměrná matice (například krychle dat)

TensorBoard – ladící nástroj pro sledování postupu trénování v grafickém rozhraní

TensorFlow – framework pro strojové učení a umělou inteligenci od společnosti Google

TensorFlow graf – set operací prováděných s daty a samotných dat

TFRecord – formát binárního souboru určeného k sekvenčnímu čtení velkého množství dat

Trénování modelu – úprava klasifikační vrstvy modelu tak, aby klasifikovala požadované třídy

XML – formát přenosu dat, jednoduše srozumitelný pro člověka i pro stroj

1 Úvod

Virtuální a rozšířená realita je novým trendem v oblasti výpočetní techniky. Aplikace zaměřené na rozšířenou nebo virtuální realitu zažívají velký rozmach, ať již v oblasti zábavního průmyslu, vývoji reálných zařízení, výuce nebo v medicíně. Využití rozšířené reality přináší nové možnosti zejména v tom, že objekty nemusí být pouze náčrty ve virtuálním světě, ale je možné si tyto objekty zobrazit přímo na reálném místě, kde se ve výsledku budou nacházet.

Dříve byla rozšířená realita dostupná pouze velkým společnostem, například automobilovým firmám při vývoji vozidel. Velké množství lidí dnes však již vlastní chytrý mobilní telefon, tablet nebo počítač s dostatečným výkonem. Díky zvyšování výkonu těchto zařízení a zlepšování kvality a přesnosti senzorů, i v levnějších zařízeních, dochází k rozšíření rozšířené reality i mezi běžnou veřejnost.

Rozšířená realita je stále více využívána v oblasti herního průmyslu. Hry pro rozšířenou realitu lze kombinovat s využitím geolokace, jako tomu bylo například u úspěšného titulu Pokémon GO. Tyto hry využívají reálné prostředí světa. Mapou hry se stává reálná mapa světa a reálné body zájmu se stávají elementy hry. Například ve výše zmíněném titulu Pokémon GO je někde na reálné mapě v okolí hráče pokémon, kterého, pokud chce hráč chytit a získat do své sbírky, tak musí najít. Najde ho tak, že půjde podle aplikace reálným světem až na místo, kde se pokémon nachází a hodí na něj pokéball. [1]

Další hry pracující s rozšířenou realitou jsou například sci-fi hra Ingress, obdoba Pokémon Go s dinosaury DinoMess, nebo Friendly Fire!: hra s ochranou základny podobná Clash of Clans, s tím rozdílem, že se odehrává v reálném světě. Dalším příkladem je ekonomická simulace Resources. Garfield GO jako další obdoba titulu Pokémon GO a mnoho dalších. [2]

Jedním z příkladů použití rozšířené reality v designerství a obchodě je aplikace Ikea Places. Ikea Places je aplikace od prodejce nábytku Ikea, určená k procházení katalogu s možností vyzkoušet si produkt přímo v prostoru vlastního bytu. Každý produkt má své rozměry a aplikace na základě dat ze senzorů počítá, jaké jsou přibližné vzdálenosti v prostoru, a následně do něj umístí daný kus nábytku. [3]

Práce bude rozdělena do několika částí. Nejprve bude vysvětleno, co je vlastně rozšířená realita a čím se liší od virtuální reality. Poté bude provedena rešerše dostupných frameworků pro rozpoznávání objektů a rozšířenou realitu. Následovat bude úvod do neuronových sítí, které se často používají ke klasifikaci obrazu. V další části bude provedena analýza požadované funkcionality výsledné aplikace a návrh jejího řešení. Předposlední kapitola se bude týkat implementace řešení pro rozpoznávání štítků ve vybraném frameworku a tvorbu aplikace pro operační systém Android, která bude vybraný framework využívat. V poslední části práce bude zhodnocení výsledků implementované aplikace a doporučení pro možné pokračování práce.

Praktická část práce je úzce provázána s teoretickou částí. Z důvodu logických souvislostí a návazností některých teoretických částí na vybrané řešení, jsou některé teoretické pojmy vysvětlovány až v praktické části práce.

2 Cíl práce

Cílem práce je provést analýzu existujících řešení pro rozšířenou realitu v mobilních aplikacích pro operační systém Android. Z existujících řešení některé vybrat a vytvořit vzorovou aplikaci využívající tento framework k rozpoznávání štítků u školních učeben v budově J. Druhým cílem aplikace je, aby dokázala rozpoznat, v které části snímaného obrazu se štítek nachází a tuto oblast označit.

Výsledkem by mělo být řešení využívající framework pro rozšířenou realitu a rozpoznávání obrázků. Toto řešení má být implementováno v mobilní aplikaci pro operační systém Android. Výsledek této práce může být následně použit v mobilní hře, zaměřené na získávání radiových fingerprintů, která může využívat výsledky této práce k určení umístění štítku pro zobrazení herního objektu a rozpoznání konkrétního štítku k určení lokace hráče.

3 Dostupné knihovny a frameworky

Díky stále stoupajícímu zájmu o rozšířenou realitu vzniklo již mnoho softwarových frameworků, které se rozšířenou realitou zabývají. Pro tuto práci jsou důležité zejména frameworky, které lze použít v mobilní aplikaci pro operační systém Android. S některými z nich seznamuje následující část práce.

3.1 Virtuální realita

Za virtuální realitu lze považovat pohled nebo interakci uživatele s počítačem vytvořeným prostředím. Virtuální realita je často spojována s tím, že uživatelé nosí speciální helmy a rukavice. Nemusí to však tak nutně být. Jednoduchá projekce na velkou obrazovku je také jistou formou virtuální reality. [4] V dnešní době se však o virtuální realitě nejčastěji mluví právě ve spojení se speciálními brýlemi pro virtuální realitu nebo celým headsetem.

3.2 Rozšířená realita

Pod pojmem rozšířená realita si můžeme představit reálné prostředí, rozšířené o nějaké virtuální objekty. Tyto objekty pak mohou být, zejména ve hrách, představovány pomocí interaktivních prvků.

Největší uplatnění má rozšířená realita na snadno přenosných zařízeních, nebo speciálních brýlích pro rozšířenou realitu. Předpokladem pro rozšířenou realitu je propojení virtuálního prostředí s reálným. Je tedy nutná jakási průhlednost displeje. Té se docílí pomocí zadní čočky fotoaparátu, která snímá reálné prostředí za telefonem. Tento obraz slouží jako podklad pro virtuální prostředí. Na základě analýzy tohoto podkladu dochází k vytvoření virtuálního prostředí, které se překreslí do vrchní vrstvy před nasnímané reálné prostředí. Tím se dosáhne cíleného efektu, kterým je zobrazení virtuálních objektů v reálném světě.

„Cílem rozšířené reality je zjednodušování života uživatelů přinesením virtuálních informací nejen k bezprostřednímu okolí, ale také k jakémukoliv nepřímému pohledu na prostředí reálného světa (jako například video live stream).“ [5]

3.3 Rozdíl mezi rozšířenou a virtuální realitou

Rozšířená realita je, na rozdíl od virtuální, založena na využití kamery zařízení ke snímání reálného prostředí, nebo na využití lokalizačních služeb, které umožňují určit přesnou polohu uživatele v prostředí. To se využívá zejména u aplikací zaměřených na venkovní lokalizaci, jako například u hry Pokémon Go.

Základním rozdílem mezi rozšířenou a virtuální realitou je tedy snímání okolního prostředí. Ve virtuální realitě dochází k tomu, že za pomoci různých

senzorů pohybu dochází k přenosu těchto vjemů do zařízení, které je schopno tyto vjemy přijmout a zpracovat. Ve speciálních aplikacích, které umožňují ovládaní pomocí virtuální reality pak dochází k tomu, že umístění a akce pozorovatele jsou řízeny na základě těchto vjemů.

Druhým rozdílem z hlediska používání, je rozdíl v tom, co při používání uživatel vidí. Při použití rozšířené reality vidí okolní prostředí s nějakými objekty navíc, kdežto ve virtuální realitě je naopak cílem, aby okolní prostředí do zážitku zasahovalo co nejméně. Dochází zde tedy k co největšímu útlumu okolního zvuku a obrazu, pro co nejvěrohodnější zážitek z pocitu virtuálního prostředí.

3.4 Frameworky pro rozšířenou realitu

3.4.1 Vuforia

Jedním z často využívaných frameworků pro rozšířenou realitu je právě Vuforia. Umožňuje implementaci rozšířené reality pro operační systémy Android, iOS a UWP aplikace pro operační systém Windows. Vuforia je placený framework, ale při dodržení určitých podmínek ji lze s určitými omezeními provozovat i zdarma.

Vuforia je v základu integrována do Unity, nebo ji lze doinstalovat. Díky této integraci je docíleno snazšího vývoje aplikací a her využívající tyto dvě technologie dohromady. [6]

Framework Vuforia má mnoho funkcí. Mezi nejdůležitější patří rozpoznávání 3D objektů podle jejich tvaru, rozpoznávání obrázků, vícerozměrných objektů podle jejich vzhledu z různých úhlů, rozpoznávání objektů a značky takzvané VuMarks. [6]

Rozpoznávání 3D objektů podle tvaru předpokládá, že objekty se nepohybují, jsou vícebarevné. Dále je důležité, aby model byl dostatečně komplexní a přesně odpovídal reálnému objektu. Reálný objekt tedy nesmí být deformovatelný. Vuforia sice slibuje rozpoznání objektu i při určité odchylce od modelu. Se vzrůstající odchylkou také vzrůstá neúspěšnost detekce.

Rozpoznávání obrázků je funkcí frameworku Vuforia, která slouží k rozpoznávání 2D objektů. Z fotografií objektů lze sestavit databázi objektů, která poté slouží frameworku Vuforia k jejich rozpoznávání.

Rozpoznávání 3D objektů je velmi podobné rozpoznávání obrázků. Rozdíl je v tom, že u 3D objektu se pro identifikaci objektu využívá více obrázků zároveň. Konkrétně lze tuto funkci využít například pro identifikaci různých krabic, které mají na každé stěně jiný obsah.

VuMark značka je ve své podstatě velmi podobná čárovému kódu nebo QR kódu. Rozdíl mezi VuMark a detekcí obrázků je zejména v tom, že VuMark značky mohou sloužit i k identifikaci dvou stejně vyhlížejících objektů. Rozdíl oproti čárovým a QR kódům je ve vzhledu VuMark značek, který je téměř neomezený, protože závisí na fantazii tvůrce. [6]

3.4.2 EasyAR

EasyAR je framework pro rozšířenou realitu od společnosti VisionStar Information Technology (Shanghai) Co., Ltd. EasyAR má dvě základní implementace:

1. EasyAR CRS (Cloud Recognition Service), jakožto webová služba určená pro vysoce škálovatelné databáze objektů, které by se špatně udržovaly přímo v zařízeních, která aplikaci vytvořenou za použití tohoto frameworku budou provozovat.
2. Druhou implementací je EasyAR SDK, sloužící k samostatné implementaci vývojářem aplikace.

Mezi funkcionality EasyAR patří API pro jazyky C, C++, podpora pro Android, iOS, Windows, Mac OS, podpora Frameworku Unity 3D, podpora skenování QR kódů, HW akcelerace, sledování obrázků, sledování více objektů najednou, sledování 3D objektů, detekce a sledování objektů různých typů zároveň a funkce SLAM[1].

EasyAR má v případě použití implementace SDK i verzi zdarma, která oproti placené verzi postrádá některé pokročilejší funkcionality (například funkci SLAM a sledování 3D objektů). [7]

[1] Funkce pro sledování pozice kamery

3.4.3 ARToolkit

ARToolkit je multiplatformní opensource framework pro rozšířenou realitu. Mezi jeho klíčové vlastnosti spadá modulárnost kódu pro snadnější použití. Podpora všech nejrozšířenějších platforem. Pro operační systémy iOS, Android, Windows, Linux a MacOS existují zkompilovaná SDK připravená k použití v aplikacích využívajících ARToolkit. Zaměření na mobilní telefony – podpora OpenGL ES2.x, integrace s GPS a kompasem a automatická kalibrace kamery. ARToolkit obsahuje také podporu pro použití v chytrých brýlích. [8]

Podporované funkce jsou sledování objektů, silná podpora kalibrace fotoaparátu, nepřerušované sledování a podpora dvojí kamery. Podpora více jazyků. Framework je samozřejmě optimalizován pro mobilní zařízení a je možné ho použít v aplikacích naprogramovaných v Unity3D.

3.4.4 TensorFlow

TensorFlow je opensource softwarová knihovna pro numerické výpočty používající grafy toku dat. TensorFlow bylo původně vyvíjeno vědci a inženýry pracujícími ve firmě Google v takzvaném Google Brain týmu ve výzkumné skupině Googlu pro vývoj strojové inteligence a strojového učení. Systém byl vyvinut pro účely strojového učení a výzkumu neuronových sítí. Návrh TensorFlow je však dostatečně obecný pro možnost použití ve velké škále jiných oborů. [9]

Mezi společnosti, které využívají TensorFlow se řadí velikáni jako NVidia, Airbnb, Intel, Google, Xiaomi, Qualcomm, Twitter, ebay, SAP a další. [9]

TensorFlow má širokou podporu opensource komunity, díky které má veliké množství podporovaných funkcí. Oproti předchozím frameworkům to není čistě framework určený pro rozšířenou realitu. Má aplikace jak v rozšířené realitě (rozpoznávání objektů z 2D obrazu) tak i v oblastech neuronových sítí, rozpoznávání zvuků, řešení matematických rovnic, OCR a dalších.

3.4.4.1 Graf toku dat

Graf toku dat je objekt, sloužící k znázornění závislostí mezi jednotlivými operacemi – k znázornění toků dat mezi jednotlivými uzly grafu. V programování se tento objekt využívá v paralelním zpracování dat. Jednotlivé uzly grafu reprezentují

výpočetní jednotky a hrany mezi uzly reprezentují data, která vstupují do uzlu nebo z něj vystupují ven. [10]

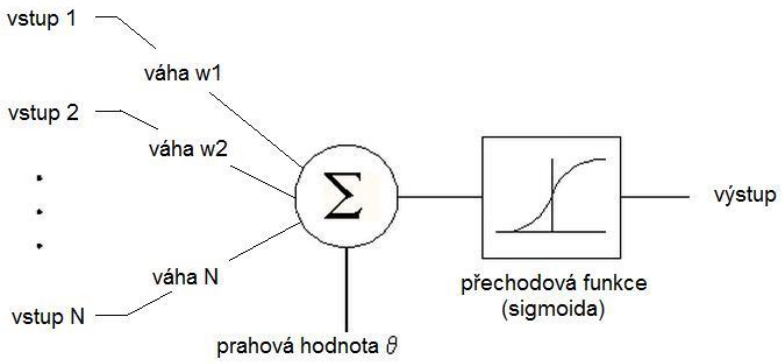
V TensorFlow to funguje tak, že nejprve je nadefinován graf toku dat a poté je vytvořeno TensorFlow sezení ke spouštění částí grafu na jednotlivých procesorech pro zajištění paralelního zpracování. [10]

3.5 Neuronové sítě

Při práci s TensorFlow se velmi často využívá model umělých neuronových sítí. Struktura neuronové sítě je podobná struktuře grafu toku dat. Jedná se o výpočetní model, který vychází z fungování biologických struktur.

Z neurologického výzkumu vyplývá, že lidský mozek je informace zpracovávající síť tvořená komplexním vzájemným propojením obrovského množství základních jednotek (neuronů). Tento systém je komplikovaný, nelineární, neurčitý, a má velmi paralelní mechanismy zpracování. Každý neuron je jednoduchá výpočetní jednotka, která se nachází v nějakém stavu, určeném nastavením samotného neuronu i okolním prostředím. Zároveň má každý neuron konkrétní mechanismus, kterým zpracuje vstupní informace na výstup. [11]

Umělá neuronová síť je založená na stejném principu. Skládá se z umělých neuronů, které jsou vzájemně propojené a předávají si signály, které transformují pomocí určených přenosových funkcí. Každý neuron má vždy několik vstupů, ale pouze jeden výstup. Vstupy jsou buď podněty z vnějšího okolí, nebo výstupy z jiných neuronů. Každý vstup má určenou váhu tohoto vstupu. Každý neuron má pak určenou prahovou hodnotu, při jejímž překonání je aktivována přechodová funkce a vygenerován výstup. [12] Obrázek 1 zobrazuje model umělého neuronu.



Obrázek 1 Model umělého neuronu [12]

3.5.1 Učení neuronové sítě

„Cílem učení neuronové sítě je nastavit váhy modelu neuronové sítě tak, aby vytvářely správnou odezvu výstupního signálu na daný signál vstupu“. [12]

Existují dvě metody učení neuronové sítě:

1. Učení s učitelem
2. Učení bez učitele

Při učení s učitelem se používá zpětná vazba. Cílem je co nejpřesněji síť naučit rozpoznávat vzor. Při učení s učitelem se tedy používá vzorů jako kontrolního mechanismu. Aktuálnímu stavu sítě je na vstup poskytnut vzor a zjištěn výsledek. Z určeného a požadovaného výsledku se zjistí chyba. V průběhu učení dochází ke změnám vah a prahů u jednotlivých neuronů, s cílem snížit hodnotu chyby. V principu učení bez učitele k této kontrole nedochází. [12]

Trénovací data se při učení rozdělují do tří skupin – učební, ověřovací a testovací. Na začátku jsou nastaveny váhy celé sítě na nějakou hodnotu (buď náhodnou nebo pro všechny vstupy stejnou) a v průběhu učení dochází k úpravám těchto hodnot. [12]

3.5.2 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou specializovaným typem neuronových sítí, určeným ke zpracování dat, která mají formu mřížky (2D) [13]. Z toho důvodu se využívají v oblasti zpracování obrazu. Konvoluce emuluje stimul neuronu na vizuální podnět. Každý neuron zpracovává pouze data z oblasti, která mu náleží. [14]

Při zpracování obrazu je to tak, že první vrstvou na vstupu je celý obraz, kde každý pixel je jeden neuron, další vrstva dostane na vstupu několik těchto neuronů, dochází tedy k segmentaci obrazu na menší části. V každé následující vrstvě se pracuje s kombinací výsledků z těchto malých částí. Dojde k tomu, že obraz je rozsegmentován na malé části, na ně je aplikován konvoluční filtr. Výsledkem celé sítě je poté vektor obsahující skóre jednotlivých klasifikovaných tříd. [14]

3.5.2.1 Pooling

V konvolučních neuronových sítích se používá metody sdružování (pooling) pro zvýraznění charakteristických rysů dat. Sdružování funguje tak, že na vstupní data se aplikuje maska o nějakém rozměru (například 2x2) a s daty na vstupu se provede nějaká operace jejíž výstupem je jedna hodnota (tedy 1x1). [15]

Používá se max-pooling a avg-pooling. Max-pooling, funguje na principu hledání maximální hodnoty, avg-pooling z dat spočítá průměr. Obě metody slouží k zvýraznění charakteristických rysů vstupu, s tím, že max-pooling je extrémnější metodou.

3.5.3 Vícevrstvé a modulární neuronové sítě

Vícevrstvá neuronová síť se skládá z více vrstev neuronů, tedy výstupy neuronů z jedné vrstvy jsou vstupy do další vrstvy. Mezi vstupní a výstupní vrstvou je tedy jedna nebo několik dalších vrstev, které si navzájem předávají data. [12]

Modulární neuronová síť se skládá z více menších sítí, které mezi sebou buď soupeří, nebo spolupracují při řešení problému. Příkladem modulární neuronové sítě je síť Inception.

3.5.4 Trénovaní sítě na vlastní data

Modely pro detekci obrazu v dnešní době mají miliony parametrů a k jejich tvorbě je potřeba obrovské množství trénovacích dat i výpočetního výkonu (stovky i více hodin práce grafické karty). Je tedy téměř nemožné vytvořit model od základu bez týmu lidí, kteří na tvorbě modelu pracují a bez speciálních serverů obsahujících několik grafických karet pro výpočty. [16]

Vzhledem k obrovské výpočetní i lidské náročnosti tvorby modelu od naprostého základu se k trénování modelu pro vlastní potřeby využívá princip přeneseného učení (transfer learning). Transfer learning je technika, která velmi snižuje náročnost na lidskou práci a zejména na výpočetní výkon díky tomu, že jako vstup využívá nějaký již natrénovaný model a model přetrénuje na vlastní data. [16]

Nevýhodou přeneseného učení je to, že výsledný model není tak kvalitní a přesný jako model, který by byl vytvořený od nuly. I přesto je však překvapivě efektivní pro mnoho aplikací [16]. Výhodou trénování pomocí přeneseného učení je,

že nevyžaduje miliony učících obrázků a na průměrné grafické kartě trvá desítky minut až několik hodin.

K trénování modelů použitých v této práci tedy bude využíváno techniky přeneseného učení, díky které bude možné otestovat více možných variant modelu a určit tak nejspolehlivější variantu.

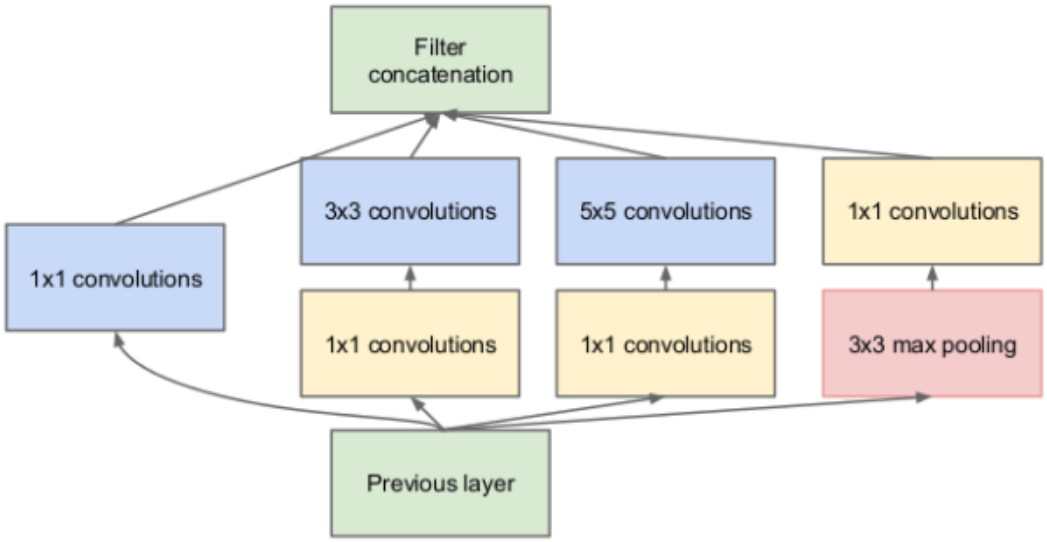
3.5.5 Inception

Model Inception byl vyvinut firmou Google jako ukázka výkonného modelu pro výzvu ImageNet Large-Scale Visual Recognition Challenge, s cílem být výpočetně efektivnější než ostatní vyzyvatelé, kteří se této výzvy účastnili [17]. Síť v této výzvě dosáhla úspěchu a stala se novým milníkem pokroku v oblasti rozpoznávání obrazu. Síť Inception přinesla dramatické zlepšení v možnostech klasifikace a detekce obrazu, a to nejen díky zvýšení výkonu hardwaru, ale také díky použití nových myšlenek a algoritmů. Důraz byl kladen také na výpočetní náročnost a spotřebu paměti. [18]

Předchozí konvoluční neuronové sítě měly typicky standardní strukturu, a to vrstvené konvoluční vrstvy (s možností doplnění o normalizaci kontrastu a max-pooling), následované plně propojenými vrstvami. Cestou, jak vylepšit tyto standardní sítě bylo jejich zvětšování, prohlubování (přidávání dalších vrstev) a rozšiřování (přidání výpočetních jednotek na vrstvu). V tomto případě však nastává problém s výpočetní a paměťovou náročností. Jako možné řešení se ukazuje vytvoření architektury, kde nebudou vrstvy navzájem plně propojené. Dnešní hardware ale není dostatečně výkonný pro zpracování řídkých datových struktur, takže zatím se nevyplatí jejich použití. Přišla na řadu otázka, jestli by nebylo možné nějakým způsobem použít aktuální hardware pro zpracování řídkých datových struktur bez ztráty výkonu. [18]

Odpověď nakonec tvůrci nalezli v literatuře zabývající se počítáním s řídkými maticemi, která naznačuje, že v případě shlukování řídké matice do menších hustých submatic je možné dosáhnout srovnatelné rychlosti výpočtů zejména při násobení těchto matic. Síť Inception těchto teorií využívá a aplikuje je do praxe. [18]

Hlavním problémem v oblasti konvolučních sítí je rozhodnutí, jaká konvoluce bude z hlediska výsledků nejlepší. Hlavní myšlenkou sítě Inception, je nechat o tom rozhodnout samotnou síť. Druhou specialitou je modulárnost, kdy celá síť se skládá z univerzálních modulů. Obrázek 2 zobrazuje strukturu jednoho Inception modulu. Každý modul sítě Inception tedy provede všechny konvoluce. V Inception byly konvoluce omezeny na 1x1, 3x3 a 5x5. Ještě se počítá max-pooling pro situaci, že by byl výhodnější operací. Výsledkem tohoto modulu je vektor, který slouží jako vstup do dalšího modulu. Konvoluce 1x1 se zde nachází pro redukci výpočetní náročnosti pomocí snížení hloubky vektoru. [18]



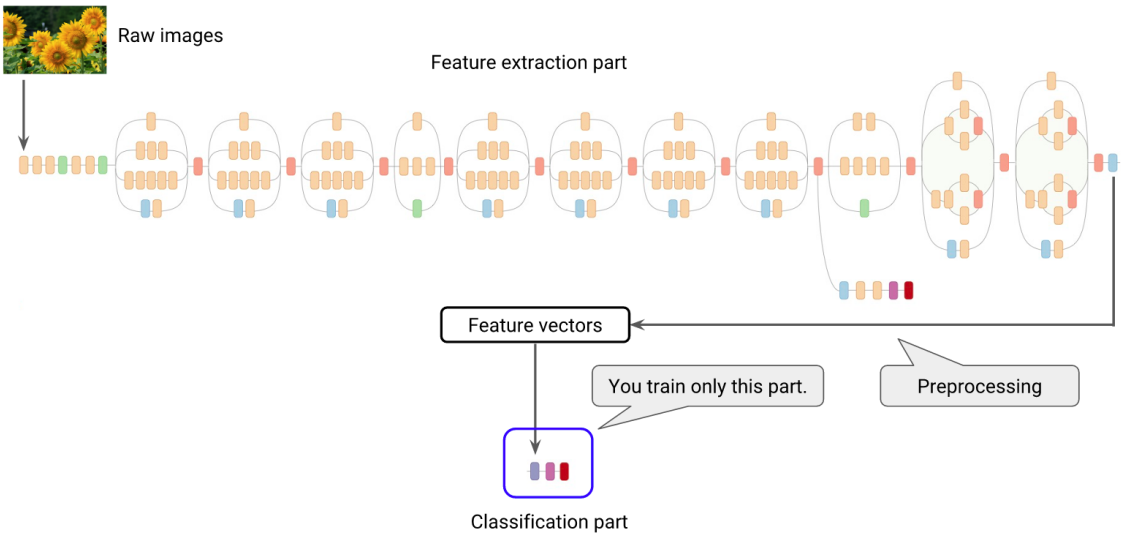
Obrázek 2 Modul sítě Inception [18]

Každá jedna výpočetní jednotka (neuron) zpracovává část obrazu. Na vstupu je obraz o rozměrech 299x299 pixelů s hloubkou 24 bitů (3 bajty – každý pro jednu barvu modelu RGB). [17]

Obrázek 3 ukazuje architekturu modelu Inception. Jednotlivé barvy bloků označují operaci, která je se vstupními daty prováděna. Žlutá barva reprezentuje konvoluci, světle modrá avg-pooling, zelená max pooling, oranžová spojení výstupů do jednoho vektoru, modrá barva označuje dropout (metoda regulace sítě, odřezává spojení mezi některými neurony). Fialová barva značí plně propojenou vrstvu, která slouží k určení třídy, do které objekt patří, a červená vsrtva je vrstva softmax (která slouží k určení pravděpodobnosti, s jakou patří objekt do dané třídy). Softmax je

matematická operace, která transformuje vektor tak aby jeho prvky patřily do intervalu (0,1) a jejich součet byl dohromady 1.

Na obrázku je možno vidět označenou část, které se týká trénování vlastních dat, tato část jsou pouze poslední 3 bloky, tedy zejména plně propojená fialová vrstva určující provázání mezi hodnotami výstupního vektoru a klasifikované třídy.



Obrázek 3 Model Inception [17]

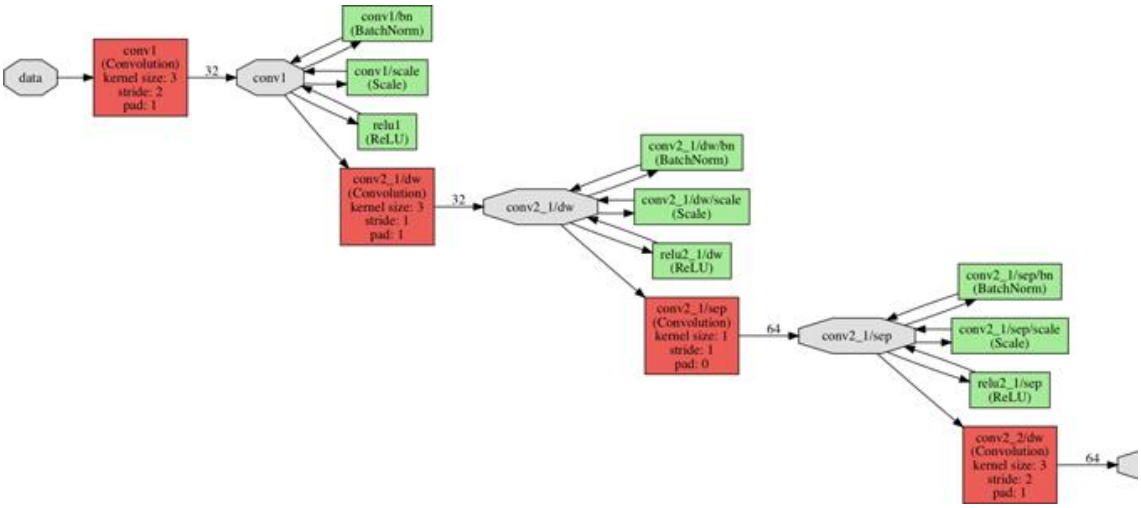
3.5.6 MobileNet

Cílem modelu MobileNet bylo vytvořit rychlou neuronovou síť. MobileNet je vysoce efektivní model určený pro mobilní zařízení. Používá datový stream s použitím depth-wise konvolucí. MobileNet sítě se primárně zaměřují na rychlost. [19]

Depth-wise konvoluce znamená, že jeden filtr se aplikuje na každý vstupní kanál, vstupní vektor je tedy rozdělen na jednotlivé kanály, na které je aplikován konvoluční filtr. Následuje point-wise konvoluce, která výsledky depth-wise konvoluce opět spojí do jednoho vektoru. Klasická konvoluce provádí obě tyto operace v jednom kroku. Díky rozdělení se snižuje výpočetní náročnost i velikost modelu. [19]

Celá síť MobileNet se pak skládá z oddělených depth-wise konvolucí, kromě první vrstvy, která je plně konvoluční. Tyto vrstvy jsou postupně poskládané za

sebou. MobileNet má celkem 28 vrstev. Obrázek 4 zobrazuje část architektury MobileNet. Je zde vidět že se střídají 3x3 a 1x1 konvoluce.



Obrázek 4 Model Mobilenet [35]

4 Analýza a návrh řešení

4.1 Analýza

4.1.1 Požadovaná funkcionalita

Cílem práce je vytvořit aplikaci, která zvládne rozpoznávat štítky u učeben na budově J Univerzity Hradec Králové a dokáže určit jejich polohu v nasnímaném obraze. Aplikace tedy musí mít implementovány 3 základní funkce.

1. Snímání okolí za pomocí čočky fotoaparátu zařízení
2. Rozpoznávání objektů v nasnímaném obraze
3. Určení polohy objektu v nasnímaném obraze

4.1.1.1 Snímání okolí za pomocí čočky fotoaparátu zařízení

Fotoaparát zařízení bude sloužit jako propojka mezi virtuálním a reálným světem. Základním předpokladem pro rozšířenou realitu bude zobrazení virtuálního objektu v reálném prostředí. Kamera zde tedy slouží k zobrazení

reálného prostředí. Zároveň se snímáním a zobrazováním reálného prostředí na displeji telefonu však kamera slouží ještě k dalšímu účelu, a to k nasnímání obrazových dat pro hledání a rozpoznávání objektů za pomoci některého z výše uvedených frameworků.

4.1.1.2 Rozpoznávání objektů

Rozpoznávanými objekty v obraze budou štítky označující učebny na budově J. Školní štítky u učeben jsou několika druhů – štítky obrazové (například šatny, WC, kuchyňka). Ukázku štítku u šatny zobrazuje Obrázek 5.

Obrázek 5 Štítek obrazový – šatna

Dále se ve škole nachází štítky textové malé (sklad, tiskárna, rozvody elektřiny), příklad takového štítku zobrazuje Obrázek 6.

Obrázek 6 Štítek textový malý (sklad, tiskárna)

Obrázek 7 zobrazuje další typ štítku, který lze nalézt v budově FIM nejčastěji, a to štítky velké s rozvrhem (štítky u učeben). Ještě se zde vyskytují menší štítky se jmény u kanceláří zaměstnanců univerzity, ukázkou takového štítku je Obrázek 8.

Obrázek 7 Štítek u učebny

Obrázek 8 Štítek malý – kancelář

Zásadním problémem u školních štítků je velmi malá rozdílnost mezi těmito štítky. Například štítky u WC a učebny se od sebe sice liší velmi, ale rozdíl mezi štítky jednotlivých učeben je velmi malý. Základním problémem, který bude použitý framework muset zvládnout vyřešit je tedy práce s velkým množstvím téměř totožných dat.

Dalším problémem je nutnost objekt rozpoznat dostatečně rychle, protože půjde o kamerový záznam. Framework tedy musí rozpoznat objekty v obraze v tak krátkém čase, v jakém to jen bude možné. Všechny výše uvedené frameworky jsou však k tomuto účelu navrženy a nemělo by jim to tedy činit větší potíže.

4.1.1.3 Určení polohy objektu

Třetím úkolem aplikace je rozpoznat, na kterém místě v nasnímaném obraze se hledaný objekt nachází. Aplikace by tedy měla umět určit souřadnice, na kterých se objekt nachází. Cílem je určení obdélníku, v kterém objekt leží.

Tento obdélník je poté možno využít v mobilní hře, pro kterou bude tato práce vzorem, jako umístění nějakého herního objektu (truhly s cenným předmětem, lékárničky, životu navíc nebo například postupu do další úrovně). Využití polohy objektu bude již záviset na fantazii tvůrce dané hry.

4.2 Návrh řešení

4.2.1 Rozpoznávání objektů

Pro rozpoznávání objektů je nutné vybrat jeden z frameworků, popsaných v části Analýza. Na základě analýzy funkcionalit jednotlivých frameworků byl vybrán framework TensorFlow. To zejména z důvodu, že díky tomu, že TensorFlow není přímo určený pro rozšířenou realitu, tak umožňuje širokou škálu analýzy obrazu. Cílem této práce je zejména co nejspolehlivější rozpoznávání štítků u učeben a efekt hry pro tuto práci není tolik podstatný.

Štítky se od sebe příliš neliší a z tohoto důvodu bohužel nelze použít framework Vuforia, který v době psaní této práce funguje na principu rozpoznávání obrázků z databáze, která je vytvořena ve webovém rozhraní. V popisované databázi jsou nahrány jednotlivé obrázky, které má Vuforia zvládnout rozpoznat. Bohužel pro každý objekt, je zde pouze jeden obrázek, což vzhledem k velmi vysoké podobnosti štítků u učeben, není možné použít.

Hlubší analýzou bylo zjištěno, že Vuforia pro rozpoznávání obrázků, potřebuje, aby se od sebe jednotlivé obrázky lišily alespoň o 30 %. Vzhledem k tomu, že jeden typ školních štítků se od sebe liší pouze čísly, napsanými na štítku, nelze Vuforii použít.

4.2.1.1 Získávání učících dat

Rozpoznávání obrázků pomocí frameworku TensorFlow vyžaduje určité množství dat, která je nutno poskytnout pro vytvoření modelu. Učením nebo také trénováním modelu se v případě této práce myslí použití některého z existujících modelů pro rozpoznávání obrázků a přetrénování jeho vrstvy na vlastní set obrázků. Nedochází tedy k definici celého modelu od nuly, ale k využití existujícího a přetrénování jeho poslední vrstvy.

K tomuto přetrénování je potřeba určité množství dat reprezentujících jednotlivé štítky. Pro každý rozpoznávaný štítek je důležité mít velké množství jeho fotografií, a to z různých úhlů, při různé intenzitě osvětlení a v různých kvalitách.

4.2.1.2 Použití TensorFlow

Po získání trénovacích dat v dostatečném množství, je potřeba tato data rozdělit do jednotlivých tříd (štítků). Pro TensorFlow Image Detection funkci je potřeba aby v každé třídě byl stejný počet trénovacích vzorků a zároveň aby pro každou třídu bylo k dispozici alespoň 30 obrázků. Se vzrůstajícím počtem kvalitních trénovacích dat dochází k růstu spolehlivosti rozpoznávání konkrétního štítku. Vzhledem k velmi malé rozdílnosti štítků je potřeba aby vzorových dat bylo dostatečné množství a v dostatečné kvalitě.

4.2.2 Určení polohy objektu v obraze

Další funkcí, kterou by měla výsledná aplikace umět je určení polohy, na které se nachází konkrétní štítek v nasnímaném obraze.

Některé frameworky pro rozšířenou realitu, jako například Vuforia, umí přímo do prostoru nějakého rozpoznaného reálného objektu zobrazit virtuální objekt. Některé umožňují i sledovat pohyb fotoaparátu 3D prostorem a podle toho správně rotovat objekt v obraze, aby nedocházelo k jeho deformacím a vypadal, že je stále na jednom místě. Z analýzy popsané výše však vychází, že framework Vuforia bohužel není vhodný na rozpoznávání málo odlišných obrazů a nelze tedy použít v námi požadované aplikaci.

Díky široké škále použití byl zvolen framework TensorFlow. Nad frameworkem TensorFlow jsou postaveny neuronové sítě, které umožňují rozpoznávání objektů a určení polohy, kde se objekt nachází. Funkce pro rozpoznávání obrázků se nazývá TensorFlow Image Detection.

4.2.2.1 TensorFlow – image detection

Image detection je funkce implementovaná v TensorFlow sloužící pro detekci obrázků. Umožňuje na základě obrazu najít, co se v obraze nachází a s jakou pravděpodobností jde o daný objekt. Tato funkce však již nemá implementováno rozpoznání přesné sekce obrázku, ve které se daný objekt nachází. Okolní objekty v nasnímaném obraze jsou tedy rušící elementy.

Tato funkce je určená pro analýzu obrázku jako celku, a lze pomocí ní tedy rozpoznávat například jestli je venku jasno nebo zataženo, jestli obraz je z města

nebo z přírody, nebo určit roční období. Cílem vyvíjené aplikace je i určit, v kterém místě snímaného obrazu se výsledný objekt nachází k čemuž slouží další funkce TensorFlow nazvaná TensorFlow object detection.

4.2.2.2 TensorFlow – object detection

Object detection je funkce TensorFlow, která umožňuje v nasnímaném obraze hledat více objektů, než jen rozpoznávat co se na obraze s jakou pravděpodobností nachází. Tato funkce již umí i ohraničit nalezené objekty a vypsat informaci, o který objekt se s nejvyšší pravděpodobností jedná.

5 Implementace

Implementace výsledné aplikace se dělí na 3 základní části. Nejprve je nutno získat data (v tomto případě fotografie) štítků u učeben, které má výsledná aplikace zvládnout rozpoznat. Po nafocení štítků v různých rozlišeních a za různých podmínek je nutné všechny fotografie rozdělit do jednotlivých tříd.

Po získání a rozdělení dat následuje nejsložitější část, a to celý proces, kterým se z fotografií rozdělených do tříd přetvoří tato data do modelu pro TensorFlow.

Vygenerovaný přetrénovaný model je poté nutné provázat do aplikace pro operační systém Android.

5.1 Získávání a klasifikace trénovacích dat

Pro to, aby bylo možné vytvořit model na základě existující neuronové sítě je nejprve nutné získat dostatečné množství trénovacích dat (v tomto případě fotografií štítků). Tyto fotografie je následně potřeba rozdělit do tříd, které budou klasifikovány.

5.1.1 Získávání dat

K získávání trénovacích dat (fotografií štítků) bude použit fotoaparát mobilního telefonu autora práce. Vzhledem k potřebě velkého množství téměř stejných fotografií bude využito nahrávání štítků ve video formátu. Pro každý štítek vznikne vlastní video, což usnadní pozdější rozdělování snímků do tříd.

Při natáčení videa štítku, bude telefonem pohybováno postupně do různých směrů i úhlů natočení směrem ke štítku. Díky čemuž bude získán snímek štítku z různých pozic pozorovatele. Po natočení videa dojde k separaci jednotlivých snímků videa do samostatných adresářů.

5.1.2 Rozdělení dat do tříd

Třída je reprezentována jako jeden konkrétní štítek. K přetrénování modelu budou tedy výsledné fotografie rozděleny do adresářů jednotlivých tříd pojmenovaných jako mXXX kde XXX je nahrazeno číslem místnosti, ke které štítek patří. V každém z těchto adresářů budou fotografie štítku z dané místnosti.

Díky rozdělení snímků z jednotlivých videí, a metodice natáčení videí tak, aby pro každý štítek bylo samostatné video, dochází k velkému usnadnění rozdělování dat do jednotlivých tříd. Po převodu video formátu do jednotlivých snímků zůstanou díky tomuto postupu v jednom adresáři fotografie jednoho konkrétního štítku.

V každém adresáři dojde k odstranění snímků s některou z následujících vad:

1. Velmi rozmazané snímky
2. Snímky bez štítku
3. Příliš tmavé nebo příliš světlé snímky

Po odmazání snímků je nutné určit, kolik snímků je k jednotlivým štítkům k dispozici a k sjednocení počtu snímků na jednu třídu. Sjednocení počtu snímků v jednotlivých třídách je nutné kvůli požadavkům technologie TensorFlow image detection. Třídy, ve kterých není dostatek kvalitních snímků budou z trénovacích dat vyjmuty.

5.2 Přetrénování modelu v TensorFlow

Pro použití TensorFlow existuje velké množství vzorových řešení jednotlivých problémů. Tato řešení lze získat na portálu GitHub. Pro zprovoznění TensorFlow API je nutné doinstalovat do PC další knihovny. Konkrétně se jedná o tyto knihovny:

- Protobuf 2.6
- Pillow 1.0

- lxml
- tf Slim
- Jupyter notebook
- Matplotlib
- Tensorflow

Pro instalaci TensorFlow je nutné mít v PC nainstalovaný Python. Na operačním systému Windows podporuje TensorFlow Python ve verzi 3.5.x a 3.6.x

TensorFlow má podporu, jak pro CPU, tak i pro GPU výpočty. Instalace verze pouze pro CPU je jednodušší a rychlejší, ale výpočty jsou mnohem pomalejší než v případě použití GPU verze.

GPU verzi API lze použít pouze v případě, že PC obsahuje grafickou kartu od společnosti NVidia s podporou technologie CUDA 3.0 nebo novější. Instalace GPU verze je o něco složitější a zdlouhavější, ale oproti CPU se vyznačuje mnohem vyšší rychlostí výpočtů.

Pro instalaci GPU verze je vhodné nejprve nainstalovat CPU verzi TensorFlow. Poté je potřeba mít v PC některý další software od společnosti NVidia a to:

- CUDA Toolkit a nastavené cesty ke CUDA toolkitu v proměnné prostředí PATH
- Ovladače NVIDIA společně s CUDA Toolkit 8.0
- cuDNN v6.0 a cesty k cuDNN doplněné do PATH proměnné
    - cuDNN (Nvidia CUDA Neural Network library) je knihovna primitiv pro podporu hlubokých neuronových sítí [20]

5.2.1 Python

Většina skriptů, které jsou připraveny vývojáři frameworku TensorFlow je napsána v jazyce Python. Tento jazyk je tedy důležitou součástí používanou při vývoji aplikací, které framework TensorFlow používají.

Python je vysokoúrovňový skriptovací programovací jazyk s licencí open-source a je tedy nabízen zdarma pro většinu používaných platforem (Microsoft

Windows, macOS, Unix) [21]. Existují dvě verze jazyka Python a to verze 2.x a verze 3.x, tyto verze jsou navzájem nekompatibilní.

„Python by šel asi nejlépe popsat jako objektově orientovaný skriptovací jazyk: v jeho návrhu se mísí jak prvky tradičních jazyků a postupů vývoje (software engineering), tak jednoduchost použití skriptovacích jazyků“ [22]

Pro provoz aplikací v jazyce Python je nutné mít v zařízení, které tyto aplikace spouští, nainstalovanou podporu jazyka Python. Podobně jako při použití jazyku Java. Programový kód se často rozděluje do modulů, které je možno doinstalovat pomocí programu pip, který se instaluje společně s podporou jazyka Python do operačního systému. Pro použití nějakého modulu ve vlastním skriptu slouží příkaz import a název třídy. Lze používat i balíčky, poté se import provádí pomocí příkazu from <balíček> import <třída>.

5.2.2 CUDA

CUDA, nebo také Compute Unified Device Architecture je technologie od společnosti NVidia, která umožňuje na grafických procesorech spouštět programy, jejichž zpracování je vhodné provádět na grafickém procesoru. Jedná se o hardwarovou i softwarovou architekturu a z toho důvodu je podporována pouze na grafických kartách od společnosti NVidia. Konkurenční společnost AMD má podobnou technologii s názvem AMD FireStream.

Softwarová část této technologie je tvořena množstvím knihoven, které jsou optimalizovány pro výpočty na grafických procesorech (GPU). Knihovny se týkají několika oblastí, jako je například lineární algebra, zpracování obrazu a videa, strojové učení, analýza grafů a další. Dostupné knihovny je možno použít v běžně používaných jazycích i ve vývojových API. [23]

Výsledné aplikace je poté možno spustit na strojích s grafickými kartami NVidia ve všech oblastech, ať už na domácím počítači, v počítačovém cloudu nebo v datacentru při použití distribuovaného zpracování. [23]

5.2.3 Základní pojmy v TensorFlow

Základním elementem pro práci s TensorFlow je objekt nazývaný Tensor. Tensor je generalizací vektorů a matic do vyšších dimenzí. Technicky se jedná o n-rozměrné pole s hodnotami tvořenými ze základních datových typů. [24]

V API TensorFlow je tento objekt reprezentován pomocí třídy tf.Tensor, která má 2 vlastnosti:

- Datový typ (float, integer, string atd.) – datový typ je známý vždy a všechny elementy v jednom tensoru mají stejný datový typ
- Tvar – počet dimenzí a velikost každé dimenze, tvar není nutné znát celý. Ve většině operací je tvar známý celý, ale existuje i možnost, kdy celý tvar je známý až v průběhu výpočtů grafu [24]

Existují také speciální druhy tensorů – tf.Variable, tf.Constant, tf.Placehodler a tf.SparseTensor. Krom tf.Variable platí, že hodnota tensoru je neměnná pro jeden výpočet. Nicméně vyhodnocení stejného tensoru může mít při každém dalším vyhodnocení jiný výsledek. Výsledkem vyhodnocení tensoru může být například náhodné číslo nebo přečtená data z pevného disku. [24]

Každý tensor má svůj stupeň. Ten je určen počtem dimenzí. Tabulka 1 obsahuje vysvětlení jednotlivých stupňů tensoru.

Stupeň tensoru	**Matematická entita**
0	Skalár (hodnota)
1	Vektor (hodnota a směr)
2	Matice (tabulka hodnot)
3	3-Tensor (krychle hodnot)
n	n-Tensor (n-rozměrný objekt hodnot)

Tabulka 1 Význam stupňů tensoru

Graf je pak objekt, který je tvořen ze struktury (operací, které se s tensory provádí) a ze samotných Tensorů, což jsou hodnoty dat, která touto strukturou prochází. Struktura grafu je určena použitým modelem.

5.2.4 Proces učení TensorFlow

Dnešní modely pro rozpoznávání objektů mají miliony parametrů a jejich učení trvá týdny. Technika pro zjednodušení tohoto problému je přetrénování nějakého již hotového modelu na nové třídy. Funguje to tak, že se vezme nějaký existující model (v TensorFlow například Inception) a přetrénuje se jeho poslední vrstva na nové třídy.

Díky využití této techniky je přetrénování modelu rychlejší a místo týdnů trvá desítky minut až hodiny na běžných počítačích. A přestože model není tak dokonalý, jako model vytvořený přímo pro konkrétní účel, tak pro většinu aplikací je dostačující.

5.2.4.1 Vytvoření tříd pro učení

Před začátkem učení je nutné mít k dispozici sadu obrázků k trénování sítě novým třídám. Pro přetrénování modelu na vlastní nové třídy je potřeba mít strom adresářů kde každý podadresář bude pojmenovaný jménem třídy a bude přímo obsahovat obrázky pro trénování modelu. Žádný adresář nesmí obsahovat obrázky z jiné třídy, v takovém případě by nedošlo ke spolehlivému přetrénování modelu. [25]

Problémy vznikající při tvorbě nejčastěji vznikají právě v datech určených pro trénování modelu. Pro správnou funkcionalitu je vhodné mít stovky obrázků pro každou třídu objektů, které je potřeba rozpoznávat. Čím více obrázků má TensorFlow při trénování k dispozici, tím spolehlivější výsledný model bude. Obrázky by také měly odpovídat výslednému použití. Pokud budou získaná data z jiného prostředí, než v kterém se bude výsledná aplikace používat, tak přesnost modelu nemusí být uspokojivá. [25]

Dalším úskalím je způsob, jakým TensorFlow funguje. V případě, že se objekty v jednotlivých třídách budou od sebe hodně lišit v barvě, pak síť vyhodnotí jako určující faktor barvu pozadí, a nikoliv strukturu objektu, která je pro rozpoznávání štítků důležitá. Pro omezení tohoto problému je vhodné data získávat za různých podmínek (v různých časech, s různým osvětlením, pomocí různých zařízení). [25]

Důležité je určit v jakém prostředí bude výsledná aplikace používána. V případě této aplikace se jedná o uzavřené prostředí a o stále ten stejný objekt rozlišený pouze svou strukturou. Odstraněním chyb ve vstupních datech lze dosáhnout velkého zvýšení přesnosti výsledného modelu.

Po vytvoření struktury dat pro trénování modelu je na řadě samotné trénování. K trénování slouží skripty (při použití TensorFlow nejčastěji skripty v jazyce Python). Tyto skripty lze získat v předpřipravené podobě na profilu TensorFlow na portálu GitHub. V těchto skriptech je potřeba upravit některé parametry a poté je lze využít.

Trénování modelu pro rozpoznávání obrázků a hledání objektů se od sebe v některých částech liší a bude popsáno v samostatných kapitolách včetně konkrétnějšího popisu skriptů, které přetrénování modelu zajistí.

5.2.4.2 Bottlenecks

V první fázi učení dojde k analýze všech obrázků v určeném učícím adresáři a výpočtu takzvaných Bottlenecků k jednotlivým obrázkům. Bottleneck je vrstva, nacházející se před finální vrstvou, která dělá klasifikaci.

Bottleneck si lze v TensorFlow představit jako set hodnot, pomocí kterých klasifikátor rozeznává obrázky od sebe (například tedy barva, struktura objektu atd.). To znamená, že bottleneck musí být dostatečně jednoznačná a srozumitelná analýza obrazu, a zároveň musí obsahovat dostatečné množství informací ke spolehlivé detekci obrazu. [25]

5.2.4.3 Učení

Po vytvoření bottlenecků pro všechny obrázky v trénovacím adresáři přejde učení do fáze 2, v které dojde k přetrénování nejvyšší vrstvy sítě na požadované obrázky. Učení probíhá v krocích. V každém kroku dojde k výpočtu přesnosti trénování, správnosti validace a křížovou entropii.

Přesnost trénování ukazuje, jaké procento obrázků z aktuálního trénovacího souboru bylo správně klasifikováno. Přesnost validace je hodnota přesnosti klasifikace určená z náhodně vybrané skupiny obrázků z jiného souboru než z trénovacích obrázků. Rozdíl spočívá tedy zejména v tom, na jakých datech je

hodnota založena. Přesnost trénování je založena přímo na obrázcích, na kterých se síť trénovala, kdežto přesnost validace je založena na jiných obrázcích. Ta tedy určuje přesnost, s jakou zvládne síť rozpoznat i jiné obrázky téhož objektu, což odpovídá výsledné funkci sítě. Je-li přesnost trénování vysoká, ale přesnost validace nízká, znamená to, že v trénovacích datech se nachází významné rysy, které však nejsou cílem rozpoznávání. [25]

Křížová entropie je ztrátová funkce úspěšnosti trénování. Cílem trénovacího procesu je tedy co nejnižší hodnota ztráty. Při úspěšném trénování by tedy s každým dalším krokem měla ztráta klesat (odchylky, kdy ztráta v některém z kroků vzroste nevadí) důležité je, že s dalšími kroky ztráta klesá. Pokud je křížová entropie moc vysoká, znamená to, že trénovací data nejsou dostatečně kvalitní a je potřebné je upravit. [25]

Bez dodatečného nastavení skriptu (skript samotný a parametry, které mu lze nastavit, budou podrobněji popsány dále v práci) běží skript 4000 kroků. V každém kroku vybere 10 náhodných obrázků z trénovacího souboru, v cache nalezne jejich vypočtené bottlenecky a pošle je do koncové vrstvy k získání odhadu o který obrázek se jedná. Odhad porovná se skutečností a podle výsledku aktualizuje koncovou vrstvu. Tímto procesem dochází ke zlepšování koncové vrstvy. [25]

Po dokončení všech kroků provede skript test spolehlivosti na obrázcích, které se procesu trénování neúčastnily a spočítá z nich výslednou přesnost klasifikace. Ta by měla být v intervalu mezi 90 a 95 procenty. Její hodnota znamená, kolik procent obrázků v testu bylo klasifikováno správně na výsledném modelu. [25]

5.2.5 TensorBoard

TensorBoard je nástroj pro jednodušší a podrobnější sledování generovaných statistik při vytváření modelu. Pomocí něj lze vizualizovat hodnoty do grafu, z kterého lze následně vyčíst, jak klesala nebo stoupala křížová entropie. Nástroj je dodáván přímo s frameworkem TensorFlow v balíčku získaném z portálu GitHub. Spouští se pomocí příkazu `tensorboard --logdir adresar_s_logy`.

TensorBoard se spustí na lokálním stroji na portu 6006. Poté, co je TensorBoard spuštěn, stačí přejít na jeho adresu v prohlížeči (localhost:6006) a v přehledném prostředí s ním lze pracovat pomocí internetového prohlížeče.

5.2.6 Využití přetrénovaného modelu pro klasifikaci

Trénovací skript vygeneruje nový model s koncovou vrstvou přetrénovanou na zadané obrázky z trénovacího souboru a k němu seznam odpovídajících tříd. Pomocí předání těchto dvou souborů do skriptu, který má za úkol rozpoznat obrázek lze klasifikovat obrázek pomocí nového modelu.

5.2.7 Python – skripty pro trénování modelu

S ostatními soubory frameworku TensorFlow lze na portálu GitHub na profilu TensorFlow nalézt i skripty, připravené pro práci s jednotlivými funkcemi, které TensorFlow podporuje. Lze zde mimo jiné tedy najít skripty pro přetrénování koncové vrstvy sítě pro detekci obrázků, ale i skripty sloužící k trénování modelu pro detekci objektů v obraze.

V této části budou jednotlivé skripty popsány, popsáno, co se v nich děje a jaké parametry do procesu vstupují. Všechny skripty jsou v jazyce Python a jsou přímo navázané na TensorFlow API. Předpokladem pro funkci těchto skriptů je počítač s nainstalovanou podporou jazyka Python, nainstalovaný framework TensorFlow a další balíčky, které TensorFlow vyžaduje.

5.2.8 TensorFlow – image detection

Pro přetrénování poslední vrstvy sítě pro detekci obrázků jsou důležité zejména tyto skripty nacházející se v adresáři image_detection:

- retrain.py – přetrénování modelu
- detect.py – klasifikace obrázků

5.2.8.1 retrain.py

Skript retrain.py slouží k přetrénování poslední vrstvy sítě na vlastní třídy z určeného trénovacího souboru dat. Tato data musí být rozdělena do podadresářů adresáře images.

Předpokladem pro spuštění skriptu retrain.py je že v adresáři images existuje samostatný podadresář pro každou třídu, kterou má výsledná síť zvládnout klasifikovat. V každém podadresáři musí být stejné množství obrázků. Obrázků musí být dostatečné množství. Obrázky by již měly být pročištěny od matoucích dat. V každém podadresáři jsou pouze obrázky odpovídající dané třídě a každý podadresář je pojmenovaný podle klasifikované třídy, kterou bude síť rozpoznávat.

Skript přijímá na vstupu tyto parametry: image_dir, output_graph, output_labels, summaries_dir, how_many_training_steps, learning_rate, testing_percentage, validation_percentage, eval_step_interval, train_batch_size, test_batch_size, validation_batch_size, print_misclassified_test_images, model_dir, bottleneck_dir, final_tensor_name, flip_left_right, random_crop, random_scale, random_brightness. Významy jednotlivých parametrů a jejich výchozí hodnoty jsou popsány níže.

- image_dir
    - cesta k adresáři s trénovacími soubory, v tomto adresáři jsou podadresáře pojmenované podle tříd a obsahují obrázky určené pro trénování modelu
    - datový typ parametru je string a nemá nastavenou výchozí hodnotu, v této práci bude používán adresář images
- output_graph
    - celá cesta včetně názvu souboru, do kterého se má uložit výsledný model
    - datový typ parametru je string a jeho výchozí hodnotou je /tmp/output_graph.pb
- output_labels
    - celá cesta včetně názvu souboru, do kterého budou uloženy názvy jednotlivých tříd
    - datový typ je string a defaultní hodnotou je /tmp/output_labels.txt
- summaries_dir
    - cesta k adresáři, do kterého se budou ukládat průběžné logy, s kterými následně pracuje TensorBoard

    - datový typ je string a defaultní hodnota je /tmp/retrain_logs
- how_many_training_steps
    - počet kroků, který se provede před ukončením tréninku
    - datový typ parametru je celočíselná hodnota a výchozí hodnota je 4000
- learning_rate
    - jak velká míra učení se používá při trénování modelu
    - datový typ float, výchozí hodnota 0,01
- testing_percentage
    - kolik procent obrázků z trénovacího souboru bude určeno pro testování
    - celočíselná hodnota, výchozí hodnota 10
- validation_percentage
    - kolik procent obrázků z trénovacího souboru bude použito pro ověřování spolehlivosti
    - celočíselná hodnota, výchozí hodnota 10
- eval_step_interval
    - jak často přepočítávat výsledky tréninku, počet kroků mezi přepočty
    - celočíselná hodnota, výchozí hodnota 10
- train_batch_size
    - kolik obrázků bude trénováno zároveň
    - celočíselná hodnota. Výchozí hodnota 100
- test_batch_size
    - kolik obrázků bude použito k závěrečnému otestování modelu. Tyto obrázky budou použity pouze jednou k otestování výsledné přesnosti nově přetrénovaného modelu. Hodnota -1 způsobí použití celé testovací sady
    - celočíselná hodnota, výchozí hodnotou je právě -1
- validation_batch_size

    - kolik obrázků se bude používat k průběžnému ověřování každých x kroků podle parametru eval_step_interval. Tato sada obrázků se používá častěji než testovací sada a slouží k průběžnému ověřování přesnosti modelu. Hodnota -1 způsobí použití celé testovací sady což u velkého množství trénovacích dat může způsobit značné zpomalení celého přetrénování modelu.
    - Celočíselná hodnota, výchozí hodnota 100
- print_misclassified_test_images
    - zda se bude vypisovat seznam všech nesprávně klasifikovaných obrázků
    - logická hodnota, výchozí hodnota False
- model_dir
    - cesta k adresáři s modelem, jehož koncová vrstva se má přetrénovat na nové třídy
    - datovým typem parametru je string s výchozí hodnotou /tmp/imagenet
- bottleneck_dir
    - cesta k adresáři, do kterého se budou ukládat vypočítané bottlenecky. Bude použit jako cache pro bottlenecky.
    - Datovým typem parametru je string s výchozí hodnotou /tmp/bottleneck
- final_tensor_name
    - název výstupní vrstvy v přetrénovaném modelu
    - datový typ je string a výchozí hodnotou je final_result
- flip_left_right
    - jestli se polovina obrázků má náhodně horizontálně otáčet
    - logická hodnota s výchozí hodnotou False
- random_crop
    - procentuální hodnota určující, jak moc se mají obrázky náhodně ořezávat
    - celočíselná hodnota s výchozím stavem 0

- random_scale
    - procentuální hodnota určující, jak moc se mají obrázky náhodně zvětšovat
    - celočíselná hodnota s výchozím stavem 0
- random_brightness
    - procentuální hodnota určující, jak moc se má náhodně měnit jas u vstupních obrázků
    - celočíselná hodnota s výchozí hodnotou 0

Po spuštění skript začne zpracovávat data. V prvním kroku proběhnou přípravy všech potřebných dat pro zpracování. Nejprve dojde k přípravě adresářů pro tvorbu logů pro TensorBoard. Následně si skript stáhne z internetu model, jehož koncová vrstva se má přetrénovat na nové třídy. Tento soubor načte a vytvoří z něj odpovídající datové struktury. Projde adresář, ve kterém se nachází podadresáře s trénovacími soubory a vytvoří si podle nastavení parametrů dané skupiny obrázků, dále také zjistí počet tříd (podadresářů). Pokud se s obrázky mají provádět některé náhodné operace (změna jasu, ořezání, zvětšení nebo otočení) tak připraví data pro tyto funkce. Následně skript přidá další vrstvu do modelu, připraví si data pro validaci výsledků a vytvoří soubor pro zápis logu. Také dojde k vytvoření bottlenecků pro všechny trénovací soubory.

Po přípravě všech potřebných dat začíná skript opakovat pro všechny kroky tyto operace: získání náhodných bottlenecků z cache, vytvoření trénovacího scénáře, a předání hodnot do modelu. Následně otestuje data z testovacího setu oproti modelu a výsledek promítne do modelu.

Po dosažení posledního kroku dojde k výpočtu finálního ověření spolehlivosti, k výpisu špatně klasifikovaných souborů a k uložení modelu do souboru na disk.

5.2.8.2 detect.py

Skript detect.py slouží ke klasifikaci obrázků podle přetrénovaného modelu vytvořeného přes skript retrain.py.

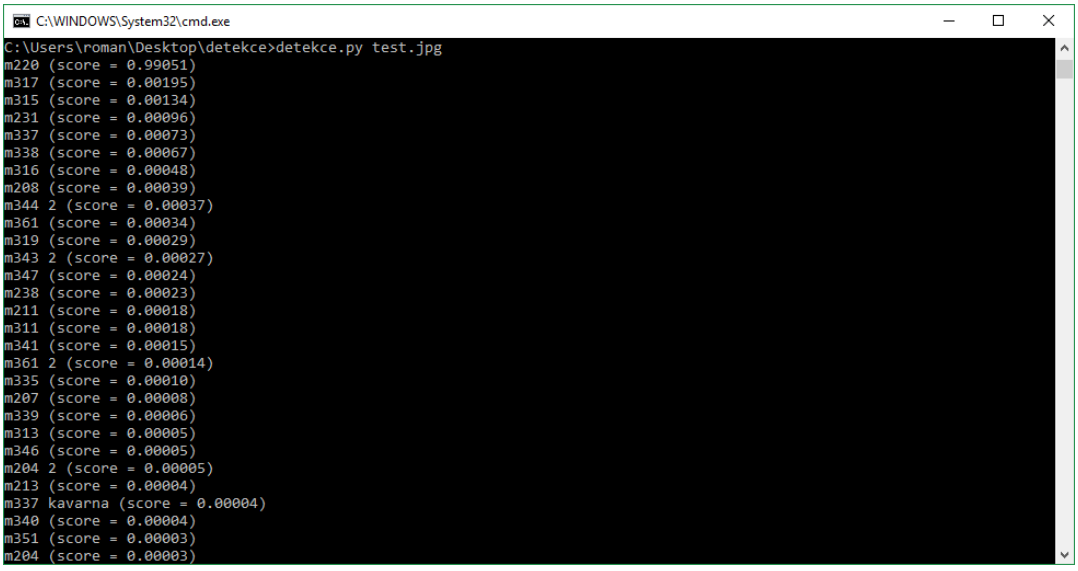
Skript si načte předaný obrázek, do cache si načte labels.txt (databázi štítků), z modelu uloženého v souboru vytvoří graf pro klasifikaci a předá dekódovaný obrázek modelu pro odhad pravděpodobnosti. Z modelu se vrátí pole pravděpodobností pro jednotlivé štítky.

Tento skript nemá žádné další parametry, které by šlo nastavit. Jediný parametr, který se tedy zadává, je cesta k souboru s obrázkem.

Volání skriptu vypadá následovně:

```
python detect.py C:\images\testovaci_stitek.jpg
```

a výsledkem je seznam pravděpodobností. Obrázek 9 zobrazuje výsledek takové detekce. Na obrázku je možno vidět, že testovací obrázek, na kterém byl štítek m220 zde má nejvyšší skóre a všechny ostatní skóre jsou nižší. To znamená, že lze prohlásit, že na daném obrázku se nachází štítek m220.



Obrázek 9 skript detekce.py

5.2.9 TensorFlow – object detection

Pro detekci objektů (object detection) je nutno připravit ze vstupních dat různá rozdílná data, která slouží jako vstup do trénovacího skriptu.

Příprava dat se dělí na několik částí:

1. Výběr obrázků, které budou určené k trénování modelu
2. Tvorba XML s určením polohy hledaného objektu
3. Vytvoření TFRecords

4. Sestavení konfigurace
5. Přetrénování modelu

5.2.9.1 Výběr obrázků pro trénování modelu

Pro trénování object detection modelu je vhodné vybrat rozmanité obrázky, na kterých je však rozpoznatelný důležitý objekt a obrázky jsou dostatečně ostré a kvalitní. Pokud se jedná o rozpoznávání nějakých objektů, které se od sebe dostatečně liší, a je pravděpodobné, že se budou na jednom snímku fotoaparátu nacházet vícekrát, je možné vybrat i obrázky (fotografie), na kterých se daný objekt nachází vícekrát. Nebo dokonce i obrázky, na kterých se nachází více různých objektů, které se budou rozpoznávat od sebe.

Množství fotografií pro trénování modelu object detection nemusí být tak velké jako u modelu pro image detection. Pro object detection není minimální počet obrázků striktně určen, a proto jich může být méně. Ovšem čím více kvalitních obrázků objektů, tím lepší výsledky daný model bude mít.

Lze tedy model postupně vylepšovat tím, že budeme přidávat, odebírat, nebo upravovat vstupní data do tréninkového modelu. Celý postup od výběru obrázků až po spuštění trénovacího skriptu je však delší než v případě image detection, a tak je vhodné připravit dostatek vstupních dat již napoprvé. Navíc samotný skript je velmi náročný na výpočetní výkon a při trénování modelu na procesoru může trénovací proces trvat i několik týdnů.

Z každé třídy bylo vybráno 20 obrázků pro trénování modelu object detection. Obrázky jsou následně umístěny do jednoho společného adresáře (na rozdíl od image detection, kde se jednotlivé třídy dělí do adresářů). Celkem máme 26 tříd, to znamená celkem 20 * 26 = 520 obrázků pro trénovací skript.

V průběhu implementace této práce pak bylo vyzkoušeno velké množství různých scénářů implementace. Tyto scénáře se liší zejména množstvím používaných dat nebo konkrétním vstupním souborem. Vzhledem k velké podobnosti není potřeba popisovat každý konkrétní scénář.

5.2.9.2 Tvorba XML pro označení objektu v obrázku

Po výběru obrázků pro každou trénovanou třídu, a jejich umístění do společného adresáře, je nutné pro každý jeden obrázek určit, na kterém místě se daný objekt nachází. To se v TensorFlow object detection modelu určí pomocí XML souboru se stejným názvem jako je název obrázku. Pokud tedy obrázek má název obrazek1.jpg pak XML soubor s určením pozice objektu se bude jmenovat obrazek1.xml. Stejný název není podmínkou, ale vhodnou konvencí pro usnadnění správy. Struktura XML dokumentu je zobrazena na obrázku 10.

```
<?xml version="1.0"?>
- <annotation>
    <folder>m213</folder>
    <filename>VID_20171110_123146 021.jpg</filename>
    <path>C:\DP_OBJECT_DETECTION\images_OD\m213\VID_20171110_123146 021.jpg</path>
  - <source>
      <database>Unknown</database>
    </source>
  - <size>
      <width>720</width>
      <height>1280</height>
      <depth>3</depth>
    </size>
    <segmented>0</segmented>
  - <object>
      <name>m213</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>3</xmin>
        <ymin>198</ymin>
        <xmax>719</xmax>
        <ymax>792</ymax>
      </bndbox>
    </object>
</annotation>
```

Obrázek 10 Struktura object detection XML dokumentu

Vytvořené XML dokumenty by měly být v PASCAL VOC formátu. PASCAL VOC je formát navržený přímo pro obrázkové datové sety pro detekci objektů. Z toho důvodu XML obsahuje i některé prvky, které se nepoužívají v případě aplikace vytvářené pro tuto práci. Formát PASCAL VOC je totiž navržen pro možnost použití různých metod detekce objektů. Projekt PASCAL VOC byl vytvářen v letech 2005 – 2012 a nyní již je kompletní. [26]

Od roku 2005 byla organizována takzvaná PASCAL VOC challenge, jejíž výsledkem byl standard pro rozpoznávání a detekci vizuálních objektů se

standardní sadou obrázků, anotací (XML popisů), a standardními procedurami pro vyhodnocování modelu detekce objektů. [26]

Tabulka 2 popisuje význam důležitých elementů v tomto XML dokumentu.

Element	**Význam**	**Příklad hodnoty**
folder	Adresář s umístěním souboru, v této práci se s ním dále nijak nepracuje	m204
filename	Název obrazového souboru, na kterém se objekty nachází	VID_20171110_123146 021.jpg
path	Cesta k obrazovému souboru	C:\DP_OBJECT_DETECTION\ images_OD\m213\ VID_20171110_123146 021.jpg
size\width	Šířka obrázku	720
size\height	Výška obrázku	1280
size\depth	Bitová hloubka (v byte)	3B = 24bitů
object\name	Název třídy objektu, jeden z nejdůležitějších elementů. Zde se očekává název třídy objektu, který se nachází v daném ohraničeném rámečku.	m213
object\bndbox	Umístění hledaného objektu v daném obraze. Všechny elementy uvnitř elementu bndbox jsou velmi důležité, protože určují rámeček, ve kterém se hledaný objekt nachází.	
bndbox\xmin	Počáteční souřadnice rámečku na ose X. Celočíselná hodnota v pixelech.	3

bndbox\ymin	Počáteční souřadnice rámečku na ose Y. Celočíselná hodnota v pixelech.	198
bndbox\xmax	Koncová souřadnice rámečku na ose X. Celočíselná hodnota v pixelech.	719
bndbox\ymax	Koncová souřadnice rámečku na ose Y. Celočíselná hodnota v pixelech.	792

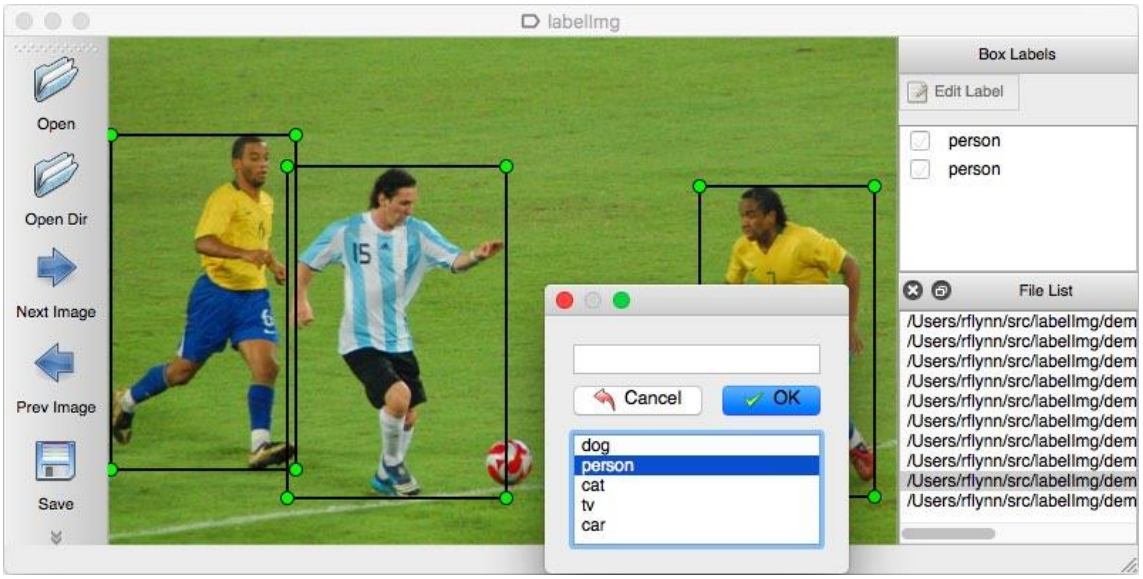
Tabulka 2 Význam elementů popisného XML dokumentu

Pro zjednodušení vytváření XML dokumentů, a odstranění nutnosti psát je ručně byla vytvořena aplikace LabelImg, což je nástroj pro anotování grafických obrázků pro object detection. Aplikace je napsána v jazyce Python s využitím knihovny Qt pro její grafické rozhraní. Anotace jsou ukládány ve výše zmíněném PASCAL VOC formátu, který je standardně používaný sítí ImageNet. [27]

„*ImageNet je databáze obrázků uspořádaná podle hierarchie WordNet (momentálně pouze podstatná jména), v níž je každý uzel hierarchie zobrazen stovkami až tisíci obrázků. V současné době máme v průměru více než pět set obrazů na uzel.*“ [28]

Aplikace LabelImg usnadňuje označování objektů v obraze, díky tomu, že do obrazu lze v grafickém rozhraní přidat rámeček, který se umístí tam, kde se nachází daný objekt. Ukázka aplikace se nachází na obrázku 11.

Aplikace tedy sama určí počáteční a koncové souřadnice na osách X i Y. Ke každému rámečku se přidá označení třídy, do které označený objekt patří a po uložení dojde k vytvoření potřebného XML souboru.



Obrázek 11 Ukázka aplikace LabelImg [27]

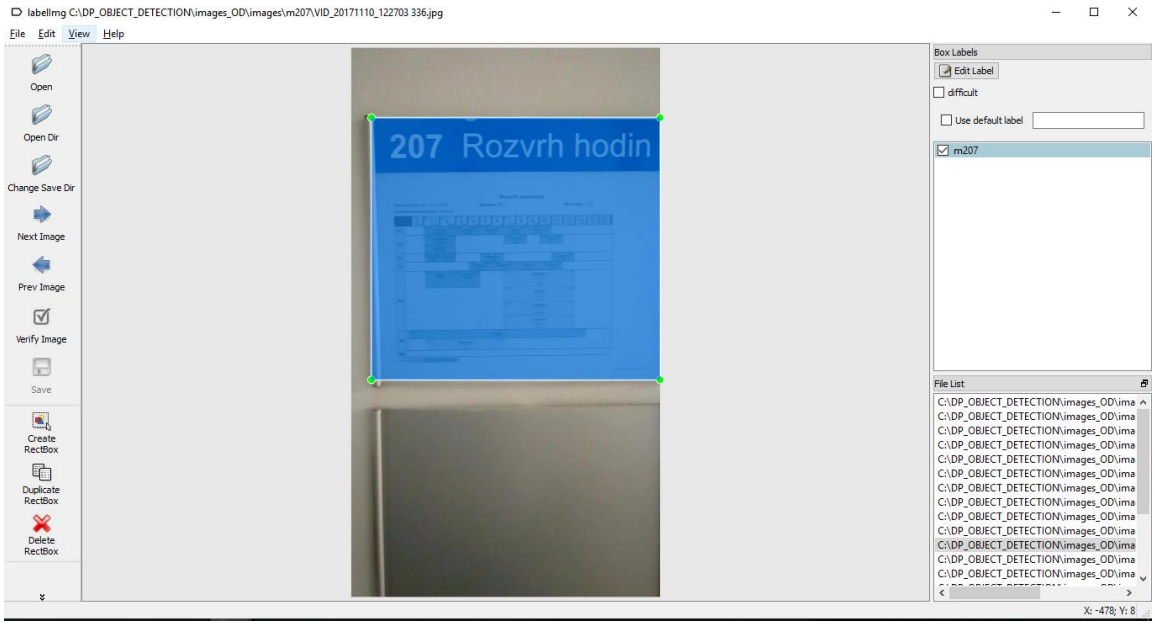
Aplikace umí pracovat i s celými adresáři, a mezi jednotlivými snímky lze přepínat pomocí tlačítek v grafickém rozhraní nebo pomocí klávesových zkratek, kterých tato aplikace pro usnadnění podporuje dostatečné množství. Tabulka 3 obsahuje seznam nejužitečnějších klávesových zkratek.

Klávesová zkratka	**Funkce**
CTRL + u	Načte všechny obrázky z adresáře
CTRL + d	Zkopírování aktuálního rámečku včetně určení třídy
W	Vytvoří rámeček
D	Další obrázek z adresáře
A	Předchozí obrázek z adresáře
Del	Odstranění vybraného rámečku
CTRL++	Přiblížení
CTRL--	Oddálení
Šipky na klávesnici	Posun označeného rámečku do směru

Tabulka 3 Seznam klávesových zkratek aplikace LabelImg [27]

Vytvořením rámečku a uložením dojde k vytvoření XML dokumentu s anotací obrázku. Při každém dalším otevření aplikace v zadaném adresáři se LabelImg

pokusí najít XML dokument odpovídající načtenému obrázku. Pokud takový XML dokument nalezne tak dojde k načtení a zobrazení rámečku i po vytvoření anotace. Lze tedy snadno XML dokument upravit bez nutnosti zásahu přímo do jeho textové podoby. Výsledek je znázorněn na obrázku 12.



Obrázek 12 Znovunačtení obrázku včetně XML anotace

5.2.9.3 Vytváření TFRecords

Po výběru souborů pro trénování a vytvoření XML anotací obrazových souborů následuje další krok, kterým je vytváření takzvaných TFRecords ze vstupního datového setu (obrázků a XML dokumentů).

TFRecords soubor obsahuje posloupnost binárních řetězců s CRC32C hashi (32 bitové CRC používající Castagnoliho polynom). Tabulka 4 popisuje formát každého TF záznamu. Formát souboru nelze číst pomocí náhodného přístupu. Je vhodný k sekvenčnímu čtení velkého množství dat, ale není vhodný k úpravám nebo dalším operacím, ke kterým se více hodí náhodný přístup. [29]

Datový typ	**Obsah**
Uint64	Length – délka dat (počet bytů)
Uint32	Masked_crc32_of_length – maskovaný CRC32C kontrolní součet délky

Byte	Data o délce length
Uint32	Masked_crc32_of_data – maskovaný CRC32C kontrolní součet dat

Tabulka 4 Struktura TFRecord

Všechny záznamy jsou spojeny dohromady a tím vytvoří soubor. Maskované CRC se vypočítá pomocí tohoto vzorce: [29]

```
masked_crc = ((crc >> 15) | (crc << 17)) + 0xa282ead8ul
```

Tvorba TFRecords souboru z dat je jedna z nejdůležitějších částí procesu. Díky tomu že data jsou v XML PASCAL VOC formátu tak je proces zjednodušený a převod XML a obrazových dat do TFRecords zajistí několik skriptů.

Před spuštěním skriptů je nutné rozdělit vstupní data set do dvou adresářů, a to do adresáře test a train. Do adresáře test bude zkopírováno přibližně 10% obrázků včetně jejich XML souborů, zbylých 90% souborů včetně XML dat bude umístěno do adresáře train. [30]

Skript xml_to_csv.py

Po rozdělení souborů přichází na řadu python skript pro převod XML anotací do CSV souborů (soubor xml_to_csv.py). Soubor lze získat na portálu GitHub[2]. Ve skriptu je potřeba nastavit cestu k adresářům test a train, změnit cestu k výstupnímu CSV souboru a přenastavit cestu k pracovnímu adresáři.

Vše, co skript provede je poté to, že projde adresáře train a test, nalezne v nich XML dokumenty a ze všech elementů object vytvoří záznam do CSV souboru. Každý tento záznam se skládá z:

- Filename – název souboru s obrázkem
- Width - šířka obrázku

[2] https://github.com/datitran/raccoon_dataset

- Height – výška obrázku
- Class – element name, název třídy objektu
- Xmin, ymin, xmax, ymax – souřadnice rámečku ve kterém se nachází hledaný objekt

Výsledný CSV soubor tedy obsahuje stejný počet řádků, jaký byl počet souborů v daném adresáři, se strukturou popsanou výše. S nastavením, které je uvedeno výše (rozdělení souborů do dvou adresářů test a train) vzniknou dva výsledné CSV soubory – test_labels.csv a train_labels.csv.

Skript generate_tfrecord.py

Druhým python skriptem, který slouží k vygenerování souborů již přímo ve formátu TFRecords souboru je skript generate_tfrecord.py. Ve skriptu je nutno doplnit funkci class_text_to_int(row_label), jejíž úkolem je mapování textového názvu třídy na číselnou hodnotu.

Pro všechny používané třídy (všechny trénované štítky učeben), je zde potřeba dodefinovat číselnou hodnotu. V této aplikaci je trénováno 26 štítků místností z FIM. Odpovídající číselné hodnoty jsou popsány v tabulce 5.

Místnost	**Třída**	**Číselná hodnota**
204	m204	1
207	m207	2
208	m208	3
211	m211	4
213	m213	5
220	m220	6
221	m221	7
231	m231	8
238	m238	9
311	m311	10
313	m313	11
315	m315	12

316	m316	13
317	m317	14
319	m319	15
335	m335	16
337	m337	17
337_kavarna	m337_kavarna	18
338	m338	19
339	m339	20
340	m340	21
341	m341	22
343	m343_2	23
344	m344_2	24
347	m347	25
361	m361	26

Tabulka 5 Mapování názvu třídy na číselnou hodnotu

Skript očekává na vstupu dva parametry:

- csv_input – cesta k CSV souboru vygenerovanému skriptem xml_to_csv.py
- output_path – cesta k výslednému souboru ve formátu TFRecords

Skript je tedy nutno spustit dvakrát, jednou pro soubor train_labels.csv a podruhé pro soubor test_labels.csv. Výsledkem budou soubory train.record a test.record.

Spuštění skriptu:

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

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

Funkcionalitou skriptu je to, že vezme CSV soubor, rozdělí data podle názvu souboru. Tím určí objekty, které se nachází ve stejném souboru. V této aplikaci je vždy na jednom obrázku pouze jeden orámovaný objekt. Pro každou skupinu objektů (pro každý jednotlivý obrázek) pak provede načtení obrázku. Zároveň načte data z XML jako název třídy, souřadnice rámečku okolo objektu (minX, maxX, minY, maxY). Data získaná z obrázku a data získaná z XML dokumentu pak zakóduje do formátu TFRecord záznamu a výsledek v binární podobě zapíše do výstupního souboru.

5.2.9.4 Sestavení konfigurace pro trénovací skript

Další součástí, kterou je potřeba vytvořit před spuštěním trénovacího skriptu object detection je konfigurační soubor, ve kterém se nastaví další hodnoty potřebné pro tento skript. Trénovací skript požaduje na vstupu obrazová data, soubory ve formátu TFRecords, původní model, jenž bude přetrénován, a právě konfigurační soubor.

Na portálu GitHub v repositáři TensorFlow lze nalézt vzorové konfigurační soubory pro různá použití. Různé konfigurační soubory jsou určeny pro rozdílné modely a liší se zejména rychlostí a přesností klasifikace. Aplikace popisovaná v této práci bude provozována na chytrém mobilním telefonu a je tedy vhodné použít model MobileNet.

Zároveň s konfiguračním souborem lze získat i samotný model v GZip formátu, ten je další součástí potřebnou jako vstup do trénovacího skriptu. Po stažení ze serveru je nutné GZip soubor rozbalit do nějakého podadresáře v pracovním adresáři.

Po získání těchto dat je na řadě úprava konfiguračního souboru podle požadavků na výsledný model. V konfiguračním souboru je několik míst přímo označených tak, že je nutno je nastavit pomocí vzorového textu popsaného v počátečním dlouhém komentáři konfigurace. V tomto případě je textem „PATH_TO_BE_CONFIGURED“, což napovídá tomu, že bude potřeba nakonfigurovat cesty k některým adresářům.

Hned na začátku konfiguračního souboru je element num_classes, který je nutno nastavit na počet tříd, které se skutečně bude skript učit. V tomto případě

tedy 26. Dále je zde možné nastavit další hodnoty týkající se porovnávání jednotlivých výsledků, testu shody, změny rozměrů vstupních dat a dalších nastavení. Naprostá většina nastavení je přednastavena podle vybraného modelu a není tedy potřebné je nastavovat.

Dalším možností je nastavit kolik obrázků se bude zpracovávat zároveň (hodnota batch_size), toto ovlivní zejména využití operační paměti, pokud tedy není dostatek operační paměti v počítači, na kterém trénovací skript poběží, je vhodné tuto hodnotu snížit. Ve výchozím stavu je nastavena na 24.

Níže v konfiguraci se poté nachází místa, kde je potřeba nakonfigurovat různé cesty k adresářům. Do pole fine_tune_checkpoint je potřeba nastavit cestu k checkpointu modelu získaného společně s konfigurací.

Dále je v sekci train_input_reader nutné nastavit cestu k TFRecord souborům (tf_record_input_reader/input_path) kam se v případě této aplikace nastaví train.record a cestu k souboru s mapováním štítků (label_map_path). Tvorba souboru s mapováním štítků bude popsána dále v této kapitole.

Poté se nastaví cesty v eval_input_readeru, kde se nastaví cesta k TFRecord souboru test.record a opět cesta k souboru s mapováním štítků (tříd na číselné hodnoty). Bude to soubor ve formátu pbtxt.

Mapování tříd na číselné hodnoty – tvorba pbtxt konfiguračního souboru

Konfigurační soubor pbtxt je dalším souborem potřebným v konfiguraci pro trénovací skript metody object detection. Jeho obsahem je několik záznamů (pro každou rozpoznávanou třídu objektů jeden), s přesně opačným mapováním, než které bylo nastaveno v předchozích krocích.

Při tvorbě TFRecord souborů byly mapovány textové názvy tříd na číselné hodnoty, v souboru pbtxt jsou mapovány číselné hodnoty na textové názvy tříd. Hodnoty mezi těmito dvěma konfiguracemi si tedy musí odpovídat.

Jedna hodnota je tedy tvořena způsobem ukázaným na obrázku 13.

```
5 item {
6   id: 2
7   name: 'm207'
8 }
```

Obrázek 13 Konfigurace pbtxt souboru

Pro každou třídu je ve výsledném pbtxt souboru tato konfigurace. V případě této aplikace tedy bude odpovídat tabulce 5.

5.2.9.5 Přetrénování modelu

Po označení obrázků, vytvoření souborů TFRecord, úpravě konfigurace modelu a vytvoření pbtxt souboru jsou připravena všechna vstupní data pro zahájení trénovacího procesu.

Před zahájením procesu je nutné ještě udělat několik dalších kroků. Skripty, které slouží k trénování modelu object detection se nachází mezi soubory získanými s celým TensorFlow frameworkem. Konkrétně se skripty nachází v adresáři research/object_detection. Buď je možné tento adresář vzít a umístit někam, kde potřebuji trénování provádět, nebo lze využít přímo tento adresář a umístit připravená konfigurační a trénovací data do něj.

K trénování je tedy potřeba v pracovním adresáři mít adresář s vygenerovanými daty ve formátu TFRecord (například adresář data), pak adresář s trénovacími daty (obrázky, XML soubory) rozdělenými do podadresářů train a test (trénovací a testovací data) – například adresář images. Dále je nutné do pracovního adresáře umístit celý adresář se staženým modelem, na jehož základě byl sestavován konfigurační soubor a bude využíván technikou transfer learning k přetrénování na vlastní data. Pak samotný konfigurační soubor a adresář, do kterého se uloží výsledný přetrénovaný model (například tedy adresář training).

Skript pro trénování modelu na object detection si v průběhu práce ukládá mezivýsledky do adresáře training a je tedy možné práci skriptu pozastavit i v průběhu a použít některý z průběžně vytvořených modelů. Čím déle skript běží, tím přesnější výsledný model bude, nicméně nejvyšší nárůst přesnosti je v několika stovkách až tisících prvních kroků. Poté již přírůstky přesnosti nejsou tak velké, a dokonce mohou stagnovat.

Před spuštěním trénovacího skriptu je nutné mít nastavenou proměnnou prostředí operačního systému s názvem PYTHONPATH s cestou k frameworku TensorFlow jako hodnotou.

V případě uložení TensorFlow adresáře na disk C je tedy před spuštěním trénovacího skriptu nutno spustit v příkazovém řádku na pracovním adresáři příkaz

```
set PYTHONPATH=C:\models-master\models-master;C:\models-master\models-master\research\slim
```

Spuštění trénovacího skriptu se provede pomocí příkazu

```
python train.py --logtostderr --train_dir=./training/ --pipeline_config_path=ssd_mobilenet_v1_pets.config
```

Skriptu je možno zadat více parametrů, všechny vstupní parametry včetně jejich významu jsou popsány v tomto seznamu:

- train_dir – adresář pro ukládání mezivýsledků a výstupů skriptu
- logtostderr – zapisovat chyby do standartního výstupu
- pipeline_config_path – kompletní konfigurace, pokud je vyplněna tak se všechny konfigurace níže (trénovací proces, input reader, model) načtou z jednoho souboru
- master – název hlavního procesu odpovědného za řízení distribuce výpočtů při distribuovaném trénování
- task – ID procesu v případě distribuovaného trénování
- num_clones – počet paralelních trénovacích skriptů spuštěných zároveň
- clone_on_cpu – provádění výpočtů na více procesorech zároveň
- worker_replicas – počet replik v případě distribuovaného trénování
- ps_tasks – počet procesů v případě distribuovaného trénování na více serverech
- train_config_path – konfigurace trénovacího procesu (počet kroků skriptu, počet obrázků zpracovávaných zároveň a další nastavení jako náhodné otáčení, náhodná změna jasu a další)
- input_config_path – konfigurace InputReaderu (cesty k souborům se záznamy typu TFRecord a pbtxt konfiguračnímu souboru obsahujícímu mapování číselné hodnoty na názvy tříd)
- model_config_path – konfigurace modelu (celkový počet tříd, nastavení změny rozměrů trénovaných obrázků). Nastavení, co se má provádět

s obrazovými daty před jejich poskytnutím samotnému frameworku pro trénování.

Jak je možno vidět, tak velké množství parametrů se používá pouze v případě distribuovaného trénování na více počítačích zároveň. Při tvorbě modelu pro tuto práci se však bude využívat pouze jeden počítač a vstupní parametry potřebné pro nastavení distribuovaných výpočtů se tedy nebudou používat.

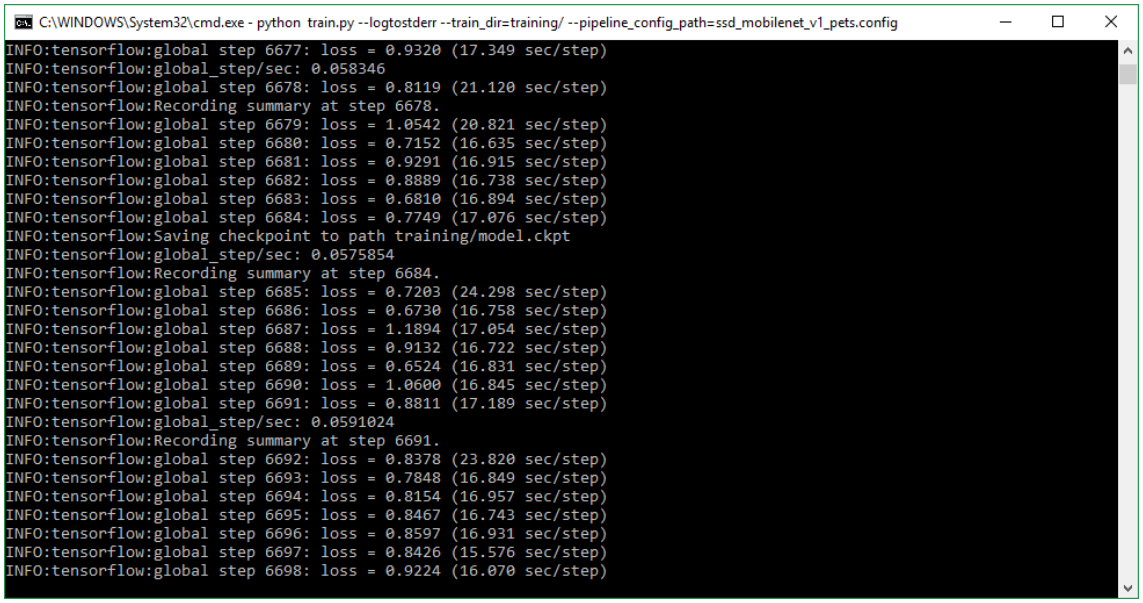
Využívat se budou primárně parametry train_dir a pipeline_config_path. Protože v tomto případě se veškerá konfigurace nachází v jednom .config souboru, tak není nutno používat parametry pro určení jednotlivých souborů obsahujících konfiguraci.

Po spuštění skript provede počáteční nastavení. Načte a nastaví si různé konfigurace podle vstupních parametrů. V případě použití distribuovaného zpracování provede nastavení těchto procesů. Nakonec zavolá metodu train ze souboru trainer.py, které předá všechna zpracovaná nastavení.

Metoda train poté provádí hlavní činnost celého trénovacího procesu. Nejprve si načte graf ze souboru a vytvoří z něj objektovou reprezentaci. Poté vytvoří konfigurace pro počítače s více grafickými kartami. V případě, že existuje nějaký postup trénování (existuje nějaký checkpoint, od kterého je možno pokračovat) tak nastaví počáteční proměnné a počáteční graf na hodnoty z daného checkpointu.

Pokud jsou v nastavení nějaké operace se vstupními daty tak je provede a poté již předá data do trénovací metody samotného frameworku.

V průběhu procesu vytváří skript checkpointy, to jsou místa, od kterých se bude pokračovat v případě přerušení skriptu. Díky tomu je také možné přerušit skript plánovaně dříve, než se provede kompletní počet kroků. Je také možné nějaký checkpoint vzít a použít v aplikaci. Tím se zjistí, jestli natrénovaný model je již dostatečně přesný.



Obrázek 14 Trénování modelu Object Detection

Obrázek 14 zobrazuje průběh trénovacího procesu. Při každém kroku skript vypíše na obrazovku, v jakém kroku se nachází, jaká je hodnota ztrátové funkce a jak dlouho krok trval. Podle nastavení pak po nějakém čase ukládá takzvané summary. Summary jsou shrnutí informací o modelu, která pak lze využít například v nástroji TensorBoard pro analýzu spolehlivosti modelu.

Dále je možné na obrázku vidět, že po kroku 6684 došlo k uložení checkpointu. Každé uložení checkpointu vygeneruje ve výstupním adresáři 3 soubory:

1. model.ckpt-<cislo_kroku>.data-00000-00001
2. model.ckpt-<cislo_kroku>.index
3. model.ckpt-<cislo_kroku>.meta

Checkpoint sám o sobě neobsahuje model, pouze stav proměnných v určitém kroku trénování. V těchto třech souborech jsou tedy definované proměnné, jejich hodnoty a informace, jak se mají tyto data mezi sebou propojit. Konkrétní funkcionalita je interním procesem celého frameworku a není tedy nutné ji znát.

Před dalším použitím checkpointu je nutné z checkpointu vytvořit TensorFlow model (graf pro odvozování – inference graph). Toho se docílí spuštěním skriptu export_inference_graph.py. Tento skript přijímá vstupní parametry popsané v tabulce 6.

Parametr	Význam
Input_type	Typ vstupního uzlu (image_tensor, encoded_image_string_tensor nebo tf_example). V případě této práce vždy image_tensor.
Input_shape	V případě, že typ vstupního uzlu je image_tensor, tak je zde možné zadat rozměry obrázku.
Pipeline_config_path	Cesta ke konfiguračnímu souboru použitému při trénování modelu
Trained_checkpoint_prefix	Cesta k checkpointu. Například training/ model.ckpt-2434
Output_directory	Adresář pro zápis výstupních souborů

Tabulka 6 Význam hodnot vstupních parametrů skriptu export_inference_graph

Výstupem tohoto skriptu je několik souborů, proto se zadává výstupní adresář a nikoli soubor. Výstupní soubory jsou transformovaný model, soubory přidružené k checkpointu a takzvaný frozen_inference_graph což je soubor, který sdružuje všechna potřebná data do jednoho souboru. V checkpointu jsou totiž hodnoty proměnných grafu (hodnoty tensorů) uložené v externím souboru. [31]

Skript tedy načte nastavení, načte vstupní soubory a přes exporter frameworku TensorFlow vyexportuje výsledné soubory.

Spuštění se provede příkazem

```
python export_inference_graph.py --input_type image_tensor
--pipeline_config_path ssd_mobilenet_v1_pets.config --
trained_checkpoint_prefix training/model.ckpt-2434 --
output_directory model_exported/
```

Po vytvoření odvozovacího grafu je možné provést ještě spuštění skriptu optimize_for_inference, který zajistí optimalizaci modelu. Vstupními parametry skriptu jsou parametry input, output, input_names a output_names. Význam těchto parametrů popisuje Tabulka 7.

Vstupní parametr	Význam
Input	Soubor s modelem, z kterého se má vytvořit výstup

Output	Soubor, do kterého se má uložit výstupní model
Input_names	Názvy vstupních uzlů. Tyto názvy lze určit v předchozích skriptech a jedná se o identifikátor poslední vrstvy modelu (vrstva natrénovaná na vlastní data)
Output_names	Názvy výstupních uzlů. Jedná se o identifikátor poslední vrstvy modelu (vrstva natrénovaná na vlastní data), který se pak používá při odvozování.

Tabulka 7 Význam vstupních parametrů skriptu optimize_for_inference

Skript provede odstranění některých operací z modelu pro snížení výpočetní náročnosti při použití ve výsledné aplikaci. Odstraní tedy operace pro ukládání checkpointu, části grafu, kterých není nikdy dosaženo a odebere operace sloužící pro ladění modelu.

Skript se tedy spustí příkazem

```
python optimize_for_inference.py –input=frozen_graph.pb --
output=optimized_graph.pb --input_names=Preloader –
output_names=final_result
```

Po dokončení běhu skriptu vznikne soubor s názvem určeným parametrem output (například optimized_graph.pb). Tento soubor pak je možno použít ve výsledné aplikaci.

5.2.10 Návrh implementace v aplikaci pro OS Android

Po natrénování modelu nastává krok otestování modelu v reálném prostředí. Aby bylo možné model otestovat, tak je nutné ho provázat do mobilní aplikace, v které bude používán. Před přidáním modelu do aplikace je nutné model optimalizovat pro použití na mobilním telefonu. Tato optimalizace zajistí odebrání některých operací v modelu, které se používaly při trénování, ale nejsou nezbytné pro klasifikaci. Tím se provede optimalizace modelu pro méně výkonný hardware v mobilních telefonech.

5.3 Implementace TensorFlow pro Android

Pro použití frameworku TensorFlow na mobilních telefonech a tabletech s operačním systémem Android, vyvinuli vývojáři frameworku několik demo aplikací. Zdrojové kódy těchto aplikací lze získat na profilu TensorFlow na portálu GitHub a následně tyto kódy upravit podle požadované funkcionality.

Zdrojové kódy se nachází na Git úložišti s celým frameworkem, a to na adrese https://github.com/tensorflow/tensorflow kde se nachází nejen zdrojové kódy samotného frameworku, ale i skripty pro použití funkcí, které TensorFlow nabízí (například image detection, object detection a další). Mimo to se zde nachází právě i ukázkové aplikace použití frameworku TensorFlow na operačním systému Android.

Ukázková aplikace se skládá ze tří menších aplikací, které všechny sdílí společné jádro a všechny pracují s videokamerou zařízení. Tyto tři aplikace jsou:

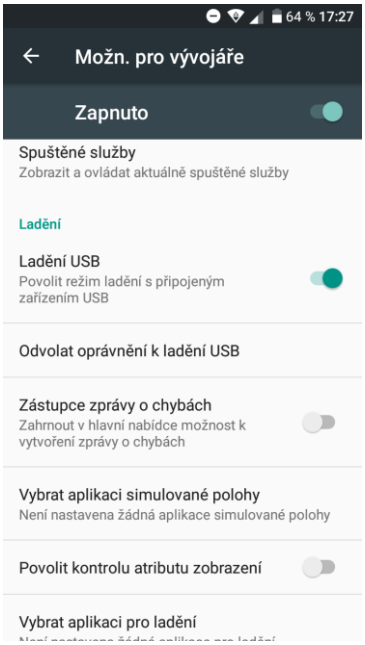
1. **TF Classify** – používá model Inception v3 k určení objektů s třídami ze sítě ImageNet. V Image netu se nachází 1000 kategorií, v nichž se nenachází mnoho každodenních předmětů a zároveň zahrnují mnoho věcí, s kterými je kontakt v reálném životě velmi nepravděpodobný. Výsledky tedy občas mohou být matoucí. Například ImageNet neobsahuje žádnou třídu Osoba, takže místo toho bude model rozpoznávat lidské bytosti jako předměty, které mají s obrázky osob něco společného (například kyslíková maska nebo bezpečnostní pás) [32]
2. **TF Detect** - používá multibox model pro kreslení ohraničujících rámečků okolo lidí na obraze. Tyto rámečky jsou popsány s hodnotou přesnosti každé detekce. Rozpoznávání nebude perfektní, protože tento typ rozpoznávání objektů je stále ve vývoji. [32]
3. **TF Stylize** – implementuje algoritmus přenosu stylů do obrazu z kamery v reálném čase. Je možno vybrat jaké styly se mají použít a zkombinovat dohromady a změnit rozlišení obrazu s kterým bude algoritmus pracovat na vyšší nebo nižší rozlišení. [32]

5.3.1 Spuštění ukázkových aplikací

Po naklonování Git repositáře do vývojového prostředí je nutno nastavit několik věcí, aby bylo možné vzorové aplikace sestavit a spustit. Nejprve je nutno mít nainstalované odpovídající Android SDK, následně je nutné upravit nastavení sestavovacího programu a poté provést sestavení a spuštění aplikace.

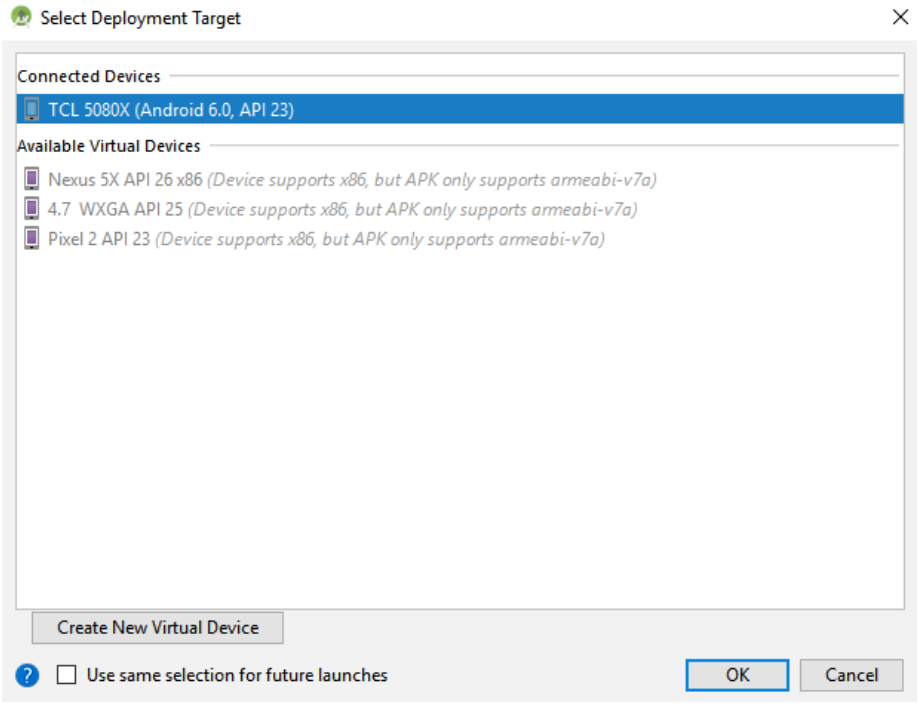
Pro provoz je potřeba mít nainstalované Android SDK ve verzi 23. V nastavení sestavovacího programu je nutné upravit hodnotu proměnné nativeBuildSystem na hodnotu none, pokud tak již není nastavená. Poté by mělo jít sestavení spustit a mělo by dojít k nainstalování všech tří aplikací do reálného zařízení. TensorFlow neumožňuje spuštění na procesorech s architekturou x86 a z toho důvodu nelze k testování použít vestavěný Android emulátor z Android Studia.

Pro spuštění na reálném zařízení je tedy nutné mít zařízení připojené přes USB a povolené ladění přes USB. To se provede přímo v zařízení přes Nastavení v sekci Možnosti pro vývojáře povolením Ladění USB. Viz nastavení na obrázku 15.



Obrázek 15 Nastavení ladění přes USB

Poté by se zařízení mělo objevit mezi možnostmi, na jakém zařízení se má aplikace spustit. Obrázek 16 zobrazuje obrazovku výběru zařízení, na kterém se má aplikace spustit.



Obrázek 16 Ladění aplikace na reálném zařízení

Po spuštění si aplikace načte konkrétní technologie potřebné k detekci, načte model a začne provádět operace s nasnímaným obrazem. Aplikace vyžaduje oprávnění na přístup k fotoaparátu (aby vůbec mohla získávat vstupní data) a k úložišti (to z ladících důvodů, protože umožňuje ukládat obraz, který vstupuje do neuronové sítě).

Všechny třídy mají za předka třídu CameraActivity, která obstarává nastavení fotoaparátu telefonu, získání oprávnění od operačního systému a získává vstupní obraz z kamery. Třída CameraActivity má implementovány i další funkce pro kompatibilitu s různými verzemi Android API a formáty vstupních dat z fotoaparátu.

V základním módu má aplikace i funkce pro zapnutí debug módu. Tyto funkce však budou z aplikace odebrány, protože bude používána pouze jedna z aplikací. Ukázku aplikace po spuštění je možno vidět na obrázku 17.

Obrázek 17 Ukázka vzorové aplikace TensorFlow pro Android [33]

Na obrázku lze vidět, že zde pomocí frameworku TensorFlow, konkrétně technologie Image detection, bylo rozpoznáno jablko Granny Smith s pravděpodobností 0,98513204 (viz modrý box na obrázku).

Toto je jedna z miniaplikací ukázkové aplikace použití frameworku TensorFlow v operačním systému Android. V tomto případě aplikace vždy rozpoznává nasnímaný objekt a na displeji telefonu zobrazuje nasnímaný obraz a seznam objektů, které na obraze mohou být, s informací o pravděpodobnosti, s jakou jde o daný objekt na základě použitého modelu.

V době rozpoznávání nepřijímá aplikace žádné další vstupní snímky, protože by došlo k zahlcení požadavky. Vždy tedy přijme jeden snímek, začne ho zpracovávat a až do uvolnění fronty zpracování všechny ostatní snímky zahazuje. Po zpracování a zobrazení výsledků opět dojde k odblokování a přijme se další snímek.

Jak lze změnit aplikaci, tak aby začala rozpoznávat vlastní objekty na základě vlastního modelu, je popsáno dále v práci.

5.3.2 Image detection

Aplikace, která se zabývá oblastí klasifikace obrázků je definována třídou ClassifierActivity. Jak je již popsáno výše, tak předkem třídy ClassifierActivity je abstraktní třída CameraActivity. Třída ClassifierActivity tedy implementuje některé abstraktní metody třídy CameraActivity. Tyto metody jsou:

- void processImage()
- void onPreviewSizeChosen(Size size, int rotation)
- int getLayoutId()
- Size getDesiredPreviewFrameSize()

Metoda onPreviewSizeChosen se volá v metodě onPreviewFrame třídy CameraActivity, ale pouze v okamžiku spuštění kamery (když ještě nebyla nasnímána žádná předchozí data) tedy pouze jednou. V ní se provádí vytvoření instance třídy TensorFlowImageClassifier, která již slouží přímo pro komunikaci s frameworkem TensorFlow.

V metodě onPreviewFrame třídy CameraActivity se také volá metoda processImage(), tato metoda již není omezena na pouze první spuštění, ale volá se pouze v případě, že aktuálně neprobíhá žádná klasifikace. V metodě processImage se v případě image detection provádí spuštění detekce ve vlastním vlákně a zároveň dochází k úpravám vstupního obrazu, zejména k ořezání do určeného rámu. Tento rám má ve výchozím nastavení rozměry 640x480 pixelů. Toto nastavení lze změnit v třídě ClassifierActivity pomocí nastavení konstanty s názvem DESIRED_PREVIEW_SIZE.

Metoda getDesiredPreviewFrameSize pouze vrací instanci třídy Size z konstanty DESIRED_PREVIEW_SIZE. Metoda getLayoutId pouze vrací odkaz na XML s layoutem fragmentu, který se má zobrazit v tomto případě `R.layout.camera_connection_fragment`.

5.3.2.1 Úpravy aplikace pro rozpoznávání vlastního setu tříd

Pro změnu aplikace tak, aby začala rozpoznávat vlastní natrénovaný set tříd stačí upravit několik konstant v definici třídy ClassifierActivity. Tyto konstanty

slouží jako konfigurace modelu. Tabulka 8 popisuje význam a hodnoty těchto konstant.

Konstanta	**Význam**	**Příklad hodnoty**
Boolean SAVE_PREVIEW_BITMAP	Určení, jestli se mají do úložiště zařízení ukládat obrázky posílané do neuronové sítě k detekci.	true
int INPUT_SIZE	Rozměr vstupních dat do neuronové sítě. Určeno na základě použité neuronové sítě. Pro Inception je to 299, pro MobileNet 224	299
Int IMAGE_MEAN	Průměrný rozdíl mezi pixely. Údaj z matematické analýzy obrazu. Je určen použitým modelem (Inception / MobileNet)	128
Int IMAGE_STD	Standartní odchylka pixelů (vypočtená na základě všech pixelů ze všech vstupních obrazů). Je určena použitým modelem (Inception / MobileNet)	128
String INPUT_NAME	Název vstupního uzlu do neuronové sítě. Určený v trénovacích skriptech.	Placeholder
String OUTPUT_NAME	Název výstupního uzlu z neuronové sítě. Určený v trénovacích skriptech.	final_result
String MODEL_FILE	Cesta k souboru s modelem	file:///android_asset/ optimized_graph.pb

String LABEL_FILE	Cesta k souboru s názvy tříd	file:///android_asset/ labels.txt

Tabulka 8 Konstanty třídy ClassifierActivity

Při potřebě používat k rozpoznávání vlastní model tedy stačí nastavit konstanty MODEL_FILE, LABEL_FILE podle názvu souborů. Hodnoty konstant INPUT_NAME, OUTPUT_NAME, INPUT_SIZE, IMAGE_MEAN a IMAGE_STD pak dohledat v dokumentaci modelu, který byl přetrénován, nebo podle trénovacích skriptů, v kterých se některé tyto hodnoty určují. Ukázku aplikace image detection lze vidět na obrázku 17.

Image detection pro tuto aplikaci nedostačuje a je tedy nutné použít i druhou část aplikace, která slouží k detekci objektů (určování polohy objektu), a to třídu DetectorActivity, která demonstruje použití funkce object detection.

5.3.3 Object detection

Aplikace zabývající se metodou object detection je definována třídou DetectorActivity. Ta stejně jako třída ClassifierActivity je potomkem třídy CameraActivity a implementuje její metody. Funkcionalita metod byla popsána v předchozí části, proto se tato část bude věnovat zejména rozdílům – tedy implementaci metod processImage() a onPreviewSizeChosen().

V metodě onPreviewSizeChosen, která je volána pouze jednou dochází také k určení základních nastavení a k vytvoření instancí detektoru a trackeru. Aplikace má podporu více detektorů, a to TensorFlow API object detection přímo v TensorFlow, MultiBox a YoloDetector. Popis jednotlivých detektorů lze nalézt v tabulce 9.

Detektor	**Popis**
TensorFlow API object detection	Detektor vyvinutý přímo pro detekci pomocí API object detection z frameworku TensorFlow. Umí použít model natrénovaný pomocí TensorFlow object detection API. Bude

	použit v této aplikaci pro svou univerzální a jednoduchou použitelnost.
MultiBox Detector	Detektor pro obecné účely založený na použití hlubokých neuronových sítí (deep neural network - DNN) podle článku Scalable Object detection using Deep Neural Networks[3]
Yolo Detector	Velmi rychlý detektor určený pro rozpoznávání v reálném čase. Detektor funguje rozdílně oproti ostatním v tom, že nepoužívá k detekci klasické klasifikátory. Jedna neuronová síť rozpoznává vše. Tento model rozděluje celý obraz na jednotlivé segmenty, na kterých pak hledá objekty a poté výsledky spojí a určí výsledek detekce. [34] Není vhodný na objekty, které se od sebe příliš neliší a tím není vhodný pro tuto aplikaci.

Tabulka 9 Popis detektorů object detection

V této aplikaci bude použit detektor založený na TensorFlow object detection API, díky její jednoduchosti použití díky dostatečně popsané dokumentaci.

V metodě processImage je pak zavolána metoda pro detekci na vybraném detektoru, a orámuje vybrané objekty. Detekované objekty s menší pravděpodobností, než jaká je určena konstantami pro jednotlivé detektory zvlášť, se nebudou vůbec rámovat. Zároveň zde funguje tracker, který vyhodnocuje výsledky detektoru, a podle nastavení odmazává nebo zobrazuje některé orámování, které nebylo detekováno. Tato funkce je tam z důvodu, aby se orámování určující polohu objektu neustále nepřekreslovalo. Orámování se

[3] https://arxiv.org/abs/1312.2249

překreslí pouze v případě, že se něco změní (například pravděpodobnost s jakou se jedná o daný objekt nebo poloha objektu.

Detekce objektů se opět provádí pouze v případě, že aktuálně jiná detekce neprobíhá. V případě průběhu detekce objektů na nějakém snímku se všechny ostatní snímky z fotoaparátu ignorují až do okamžiku, kdy je detektor zase uvolněn pro další zpracování.

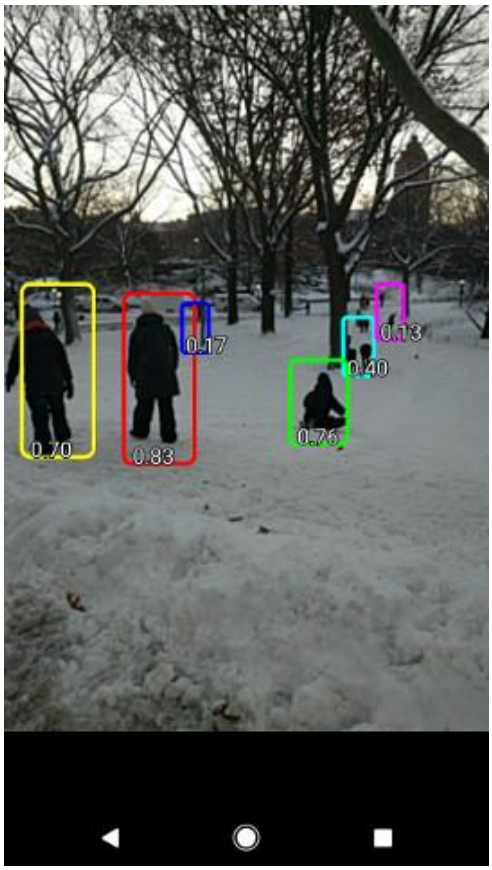
V třídě DetectorActivity je opět několik konstant, které je nutné nastavit v případě, že je třeba detekovat vlastní objekty. Tyto konstanty stačí změnit a aplikace začne detekovat odpovídající objekty. Význam konstant, které se týkají funkcí použitých v této aplikaci popisuje Tabulka 10.

Konstanta	**Význam**	**Příklad hodnoty**
int TF_OD_API_INPUT_SIZE	Velikost snímků, které prochází funkcí pro detekci objektů. Vždy je očekáván čtverec o straně délky TF_OD_API_INPUT_SIZE	300
String TF_OD_API_MODEL_FILE	Cesta k souboru s modelem	file:///android_asset/ssd_mobilenet_v1_android_export.pb
String TF_OD_API_LABELS_FILE	Cesta k souboru s názvy detekovaných tříd	file:///android_asset/coco_labels_list.txt
DetectorMode MODE	Vybraný detektor (TF_OD_API / YOLO / MULTIBOX)	DetectorMode.TF_OD_API
Float MINIMUM_CONFIDENCE_TF_OD_API	Minimální jistota (pravděpodobnost, že se jedná o daný štítek), při které se zobrazí orámování	0.6f

Boolean SAVE_PREVIEW_BITMAP	Zda ukládat snímky, které jsou posílány k detekci, na úložiště zařízení.	false
Float TEXT_SIZE_DIP	Velikost textu v DIP[4], kterým bude u orámování napsán název třídy a pravděpodobnost	10

Tabulka 10 Konstanty třídy DetectorActivity

Ukázku aplikace používající object detection zobrazuje Obrázek 18.



Obrázek 18 Ukázka aplikace object detection [33]

[4] Density Independent Pixel – virtuální pixel nezávislý na skutečném rozlišení obrazovky

Metoda object detection bude použita pro detekci polohy objektu, výsledky rozpoznání jednotlivých tříd však nejsou příliš přesné a object detection se tedy nehodí pro rozpoznávání štítků od sebe. Může být však použit na určení polohy štítku což zvládá s dostačující přesností. Proto budou ve výsledku použity obě neuronové sítě. Jedna bude detekovat polohu štítku a druhá bude klasifikovat, do které třídy štítek patří.

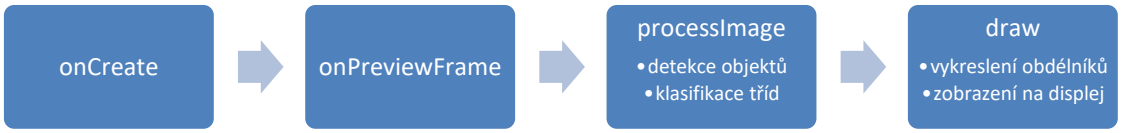
5.3.4 Spojení image detection a object detection do jedné aplikace

Z důvodu potřeby identifikovat jak polohu štítku v nasnímaném obraze, tak i rozpoznat o jaký štítek se jedná, bylo nutné spojit tyto dvě metody do jedné aplikace.

Toho bylo docíleno pomocí kombinace třídy ClassifierActivity a DetectorActivity. Nejprve bylo nutné do třídy ClassifierActivity přidat všechny konstanty, které tvoří nastavení funkcí pro object detection. Tedy vytvořit konstanty TF_OD_API_INPUT_SIZE, TF_OD_API_MODEL_FILE, TF_OD_API_LABELS_FILE. Dále bylo nutné ke klasifikátoru ještě přidat detektor a proměnné týkající se stavu detektoru. Proměnná computingDetection pro určení, jestli probíhá výpočet detekce, tracker jako instance třídy MultiBoxTracker (není to stejné co MultiBoxDetector), dále nastavit mód (konstanta MODE) na DetectionMode.TF_OD_API a doplnit konstantu MINIMUM_CONFIDENCE_OD_API pro určení minimální přesnosti, při které se začnou rámovat objekty v obraze.

Layout aplikace byl změněn na camera_connection_fragment_tracking, kam byl doplněn výpis jednotlivých klasifikovaných tříd. Do metody onPreviewSizeChosen bylo doplněno vytvoření instance trackeru (MultiBoxTracker) a detektoru (TensorFlowObjectDetectionAPIModel).

Také musela být upravena funkcionalita aplikace, postup aplikace přes jednotlivé metody lze vidět na schématu, který zobrazuje Obrázek 19.



Obrázek 19 Schéma aplikace

Dále bylo nutné do metody doplnit volání funkce onFrame na instanci třídy MultiBoxTracker pro zachování funkcionality rámování popsané výše a doplnění funkcí pro rozpoznávání objektů k funkci klasifikace. Tím bylo docíleno toho, že každý zachycený snímek se začal zpracovávat jak v modelu image detection, tak v modelu object detection.

Metoda processImage tedy provede získání výstupů z frameworku TensorFlow na základě poskytnutých vstupních dat, jimiž jsou konfigurace podle konstant a zachycený snímek z kamery zařízení. Po získání výsledků nastaví výsledky do view zobrazujícího výsledky klasifikace.

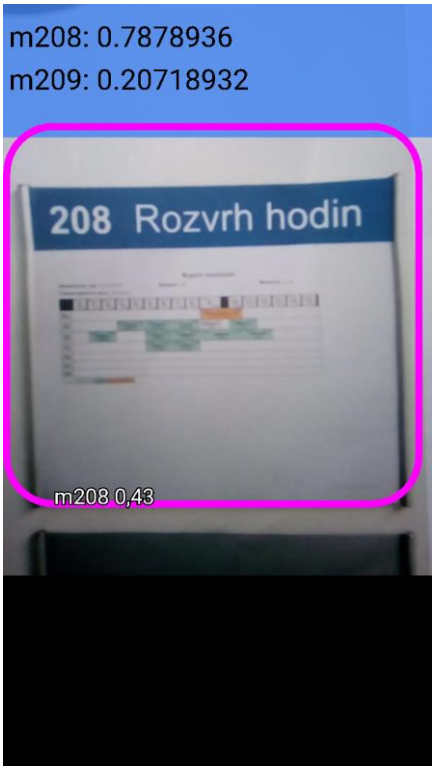
V aplikaci je vyvinuto vlastní View, které umožňuje nastavit callback, který se s obrazem provede před zobrazením. Jedná se o třídu OverlayView, která má jako předka přímo třídu View, které přepisuje metodu draw() tím způsobem, že nejprve na výstupním canvasu (grafický zobrazovaný objekt) spustí callbacky, které má nastavené.

V třídě ClassifierActivity se v metodě onPreviewSizeChosen, která se spouští pouze jednou, nastavuje callback pro zajištění funkcionality MultiBoxTrackeru a to takovou, že na instanci třídy MultiBoxTracker zavolá metodu draw a předá jí tento vykreslovaný canvas.

Metoda draw třídy MultiBoxTracker provede transformace se souřadnicemi (převede souřadnice ze vstupního obrazu do sítě do vykreslovaného obrazu). Vykreslovaný obraz je 640x480 pixelů, kdežto vstupní do modelu je 299x299 pixelů. Vstupním obrazem do sítě object detection je tedy transformovaný vstupní obraz. Obrázek 20 zobrazuje vstupní obraz do neuronové sítě vytvořený ze snímku fotoaparátu. Takový obraz vznikne transformací vstupních dat do rozměrů 299x299 pixelů. Obraz byl získán pomocí přenastavení konstanty SAVE_PREVIEW_BITMAP na hodnotu true.

Obrázek 20 Vstup do sítě object detection

Pro všechny odpovídající detekce poté aplikace vykreslí obdélník s popisem, který obsahuje název detekované třídy a přesnost detekce. Jak vypadá výsledná aplikace zobrazuje Obrázek 21.

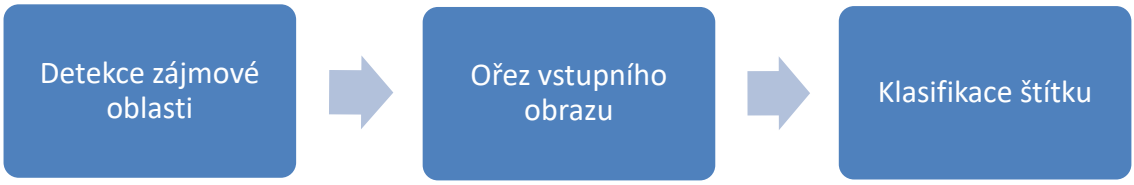


Obrázek 21 Ukázka kombinace object detection a image detection

Výsledná aplikace provádí rozpoznávání na stejném vstupním obrazu. Výsledky, které síť poskytuje na těchto datech však nejsou optimální a pro zvýšení

přesnosti rozpoznávání by mohlo být lepší nezpracovávat stejný obraz oběma sítěmi, ale nejprve detekovat polohu objektu (určit souřadnice orámování). Poté by bylo možné podle tohoto orámování oříznout vstupní obraz pouze na zájmovou oblast. Tento ořez poté poskytnout jako vstup síti, která má na starosti určení štítku. Tímto se zamezí problému s rušivými okolními elementy obrazu, které mohou síť plésti a způsobovat nejistotu. Zároveň se tím i zvýší plocha, která obsahuje číslo místnosti, které je jediným odlišením od ostatních štítků stejného typu.

Posledním krokem úpravy aplikace tedy bude operace předsunutí jedné sítě té druhé. Obrázek 22 zobrazuje výsledné schéma detekce v upravené aplikaci.



Obrázek 22 Schéma upravené aplikace

5.3.5 Detekce polohy štítku před klasifikací

Poslední úpravy aplikace – tedy zpracování obrazu tak, aby nejprve byla v obraze nalezena zájmová oblast a z ní teprve byl detekován štítek druhou sítí se docílí pomocí několika kroků.

1. Detekce zájmové oblasti
2. Ořez vstupního obrazu
3. Klasifikace štítku

Nejprve je nutné určit zájmovou oblast. Zájmová oblast je vlastně oblast v obraze, kterou určí síť pro rozpoznávání objektů (object detection). Oblast je určena orámováním, které je vytvořeno na základě souřadnic, které vygeneruje tato síť. Udržování rámečků zajišťuje třída MultiBoxTracker, které tedy je nutné poskytnout nějaká další vstupní data popsaná v tabulce 11.

Vstupní parametr	**Význam**
ResultsView resultsView	View z GUI, do kterého se mají zobrazit výsledky ze sítě pro klasifikaci štítků.

Classifier classifier	Instance klasifikátoru obrázků pro rozpoznávání, který štítek se v zájmové oblasti nachází.
Bitmap croppedBitmap	Snímek z kamery, se kterým bude klasifikátor pracovat.

Tabulka 11 Vstupní parametry do MultiBoxTrackeru po změně funkcionality detekce

Tyto tři parametry se nastaví objektu třídy MultiBoxTracker v metodě processImage, konkrétně v metodě runInBackground, která zajišťuje spuštění kódu ve vlastním vlákně.

Do metody draw třídy MultiBoxTracker je poté potřeba doplnit zpracování těchto vstupních dat. Nejprve je nutné získat orámování objektu, které bylo zjištěno, toto orámování bude uloženo do proměnné TrackedRecognition recognition1. Pro optimalizaci je zde doplněna kontrola, aby se obraz poskytoval síti image detection pouze v případě, že nějaký objekt byl v obraze nalezen nějaký objekt sítí object detection. V opačném případě se klasifikace přes síť image detection ani nebude spouštět.

Pokud nějaký objekt je nalezen pak je podle něj oříznut vstupní obraz. Zde je nutno ošetřit situaci, kdy síť object detection může určit polohu objektu i mimo souřadnice obrazu. Toto nastává konkrétně v případě, kdy je zájmový objekt v některém z okrajů, ale zároveň je dostatečně vidět a síť ho zvládne rozpoznat. Pro klasifikátor je ještě nutné vstupní obraz oříznout na čtvercový rozměr. Toho lze v jazyce Java docílit pomocí metody createScaledBitmap z třídy Bitmap, které se zadá původní obrázek a nové rozměry (výška a šířka). V tomto případě se tedy jako rozměr zadá hodnota konstanty INPUT_SIZE ze třídy ClassifierActivity.

Výstupní oříznutý obraz je předán síti image detection pro klasifikaci štítku. Tento klasifikátor vrátí výsledky, které se poté propíší do View určeného vstupním parametrem resultsView. Z metody processImage třídy ClassifierActivity je poté odebráno spouštění klasifikátoru, protože se spouští v jiné části kódu.

V aplikaci, která vznikne na základě této diplomové práce se předpokládá, že tyto výsledky se nebudou vypisovat na obrazovku, ale budou použity k určení

nějakého herního objektu. Výsledkem tedy je oblast, v které se štítek nachází, a zároveň identifikátor štítku, který síť zvládla rozpoznat na základě výřezu dané oblasti ze vstupního obrazu. Výsledná aplikace vypadá tedy stejně jako na obrázku 21, s tím rozdílem, že nyní druhá síť pracuje s jednoznačnějším vstupem, na jehož základě je možné s vyšší spolehlivostí poznat štítek, protože obraz neobsahuje nedůležité prvky.

6 Testování a výsledky

Důležitou součástí implementace aplikace bylo průběžné testování různých variant. V průběhu práce docházelo k obměnám jak aplikace, tak i modelu. V průběhu vývoje byly vyzkoušeny desítky různých variant lišících se trénovacími daty, použitou neuronovou sítí, funkcionalitou v aplikaci nebo nastavením trénovacího skriptu.

Pro otestování sítě byla vymyšlena metodika, která je popsána v následující části. Po ní následuje část, která zhodnocuje výsledky hlavních vyzkoušených variant řešení.

6.1 Metodika testování

Vzhledem k obtížnému určení spolehlivosti pouze na základě laboratorních testů prováděných v domácím prostředí autora práce, a kvůli nedostupnosti školních tabulí v laboratorním prostředí pro rychlé otestování každé varianty bylo testováno třemi způsoby.

1. Rozpoznání objektu z displeje
2. Rozpoznání objektu z tisku fotografie štítku
3. Rozpoznání objektu v reálném prostředí na budově J

Testování také probíhalo dvěma metodami. V případě větších úprav modelu bylo testování podrobnější než v případě pouze menších úprav modelu. V případě velké úpravy modelu byla výsledná aplikace, používající tento model, postupně otestována na několika fotografiích zobrazených na displeji, poté na fotografii štítku zobrazené na tisku a poté i v reálném prostředí ve škole na budově J. V případě menších úprav bylo testování prováděno pouze prvními dvěma způsoby.

Spolehlivost se v tomto případě obtížně kvantifikuje, každé řešení totiž poskytuje jiná výstupní data a mohou nastat situace, kdy aplikace zachytí rozmazaný obraz (například kvůli pohybu zařízení) a již nezvládne štítek rozpoznat. Určení spolehlivého rozpoznání se tedy i liší podle toho, která metoda se testuje.

Principem testování bylo nasměrování telefonu tak aby zabíral oblast štítku a jeho ustálení v přibližně stejné poloze bez pohybu. V okamžiku, kdy se výsledky sítě ustálily (při každé klasifikaci byly velmi podobné) tak byla odečtena hodnota spolehlivosti ukázaná na displeji a považována za spolehlivost rozpoznání daného štítku. V případě testování aplikace v laboratorních podmínkách bylo vždy tímto způsobem otestováno několik fotografií pro každý štítek). V reálném prostředí je stejný štítek vždy jen jeden.

Testování probíhalo na dvou telefonech s nastaveným rozlišením aplikace 640x480 pixelů. Prvním z telefonů je Alcatel Shine Lite 5080X, druhým použitým telefonem k testování je pak Samsung Galaxy A5 (2016). V případě, že síť neposkytovala dostatečné výsledky ani v laboratorním prostředí, tak testování v reálném prostředí nebylo provedeno. Jedná se o případy, kdy síť nezvládla z displeje ani z tisku spolehlivě poznat štítek ve více než 50% případů. V případě že síť zvládla rozpoznat štítek z displeje a z tisku více než v 50% případů (bez ohledu na odhadovanou spolehlivost) tak byla síť otestována v reálném prostředí.

Výsledky testování v laboratorních podmínkách a v reálném prostředí se od sebe částečně liší, ve většině případů platilo, že otestování v reálném prostředí mělo o něco lepší odhady spolehlivosti. To je dáno zejména tím, že zobrazením na displeji nebo tiskem je již dvakrát snížena kvalita testovaného objektu, poprvé vyfocením a podruhé nasnímáním zobrazené fotografie.

6.1.1 Rozpoznání štítku

Při určení spolehlivosti rozpoznání štítku bylo provedeno více měření. Nejprve byla síť testována na fotografii štítku zobrazeném na displeji. Na základě výsledků, které síť poskytla bylo vyhodnoceno, jaká je přibližná průměrná spolehlivost. Tedy pokud bylo otestováno 5 štítků, každý na 5 fotografiích tak výsledná spolehlivost byla určena jako průměrná maximální hodnota těchto spolehlivostí.

Rozsah spolehlivostí u štítků, které zvládla síť rozpoznat je poté považován za spolehlivost dané sítě. Pokud se tedy u většiny testovaných štítků určí spolehlivost rozpoznání v rozmezí 60 – 70% pak toto rozmezí je považováno za výslednou spolehlivost testované sítě.

6.1.2 Určení polohy štítku

U metody určení polohy štítku byla spolehlivost hodnocena na základě toho, jestli síť zvládla správně určit polohu štítku v obraze. Pokud docházelo k rámování celé obrazovky nebo docházelo k orámování jiných oblastí než oblastí, kde se štítek nachází, tak poté nebyla síť považována za spolehlivou.

6.2 Výsledky

Při vývoji aplikace, která je výsledkem této práce bylo postupně otestováno několik různých modelů s různým množstvím i kvalitou dat. Mimo popsané hlavní varianty byly testovány desítky různých podvariant těchto řešení. Popsané výsledky v sobě zahrnují i výsledky, kterých bylo dosaženo u podvariant. Uvedené hodnoty přesnosti jsou rozsahy přesnosti štítků, které se podařilo správně určit, jedná se tedy o nejlepší výsledky, kterých síť při testování v reálném prostředí dosáhla.

Testované hlavní modely byly tyto:

1. 500 obrázků pro každou třídu, model Inception + object detection pro detekci oblasti, kde se nachází štítek. Tento model byl na mobilních telefonech velmi pomalý a nespolehlivý (spolehlivé rozpoznání místa se štítkem, ale nespolehlivé rozpoznání učebny). Výsledkem: odmazat rozmazané fotografie = zvýšení kvality trénovacích dat pro zlepšení spolehlivosti modelu.
2. 380 obrázků pro každou třídu (po odmazání rozmazaných trénovacích dat), model Inception + object detection pro detekci oblasti, kde se nachází štítek. Tento model pomalý, ale spolehlivější v rozpoznání, o jaký objekt se jedná (určení štítku v reálném prostředí s přesností pohybující se v rozmezí 80 – 90%). Novým cílem se tedy stává zvýšení rychlosti detekce.

3. 380 obrázků pro každou třídu, model MobileNet pro klasifikaci učebny, object detection pro určení oblasti, kde se štítek v obraze nachází. Model postavený na síti MobileNet je na stejném mobilním telefonu přibližně 4x rychlejší než u varianty číslo 2 s modelem Inception. Problémem se však stává spolehlivost (časté chybné rozpoznávání štítků, s tím souvisí malá přesnost rozpoznání v rozmezí 50 - 60%). Dalším postupem se stává snaha naučit model na větším množství kvalitních trénovacích dat.
4. Snaha zvýšit přesnost rapidním zvýšením počtu vstupních dat pro jednotlivé třídy – zvýšení vstupních dat na 1500 obrázků ve vysokém rozlišení pro každou třídu. Přesnost modelu se zvýšila o přibližně 10%, což stále není dostatečně přesné. Novým cílem se tedy stává i za cenu zpomalení detekce přepracovat aplikaci tak, aby nejprve nalezla umístění štítku (tedy zájmovou oblast) a druhá síť klasifikovala štítek již pouze podle výřezu zájmové oblasti.
5. Aplikace po přepracování funguje nově tak, že nejprve určí umístění štítku. Poté co úspěšně detekuje nějaký štítek v obraze, se z nasnímaného obrazu udělá výřez oblasti, kde se štítek nachází a tento výřez slouží jako vstup do modelu pro určení štítku. Výsledkem je aplikace, která je sice pomalejší než při asynchronním zpracování, ale stále je rychlejší než při implementaci rozpoznávání učebny na síti Inception. Nastalo zvýšení přesnosti určení učebny. Nová nejlepší spolehlivost se pohybuje v rozmezí 75 – 85%. Zůstává zde však problém s tím, že některé třídy nezvládne síť vůbec rozpoznat, nebo je dokáže určit s přesností menší než 50%.

Výsledkem testování je informace, že zatím žádný model nedokáže s velkou přesností určit, o jaký štítek se jedná. Nejvyšší přesnost neuronové sítě postavené na architektuře MobileNet, která je dostatečně rychlá, se pohybovala okolo 85% což může již být považováno za dobrý výsledek. Problém však nastával u rozpoznávání učeben s čísly, která vypadají podobně, jako například 206 a 208.

Finální aplikace byla otestována 5krát a pro každou třídu byl vybrán nejlepší výsledek, kterého síť dosáhla. Jako minimum byla stanovena spolehlivost 0,5. Všechny třídy, které této spolehlivosti nedosáhly, byly vynechány. Z 26 testovaných

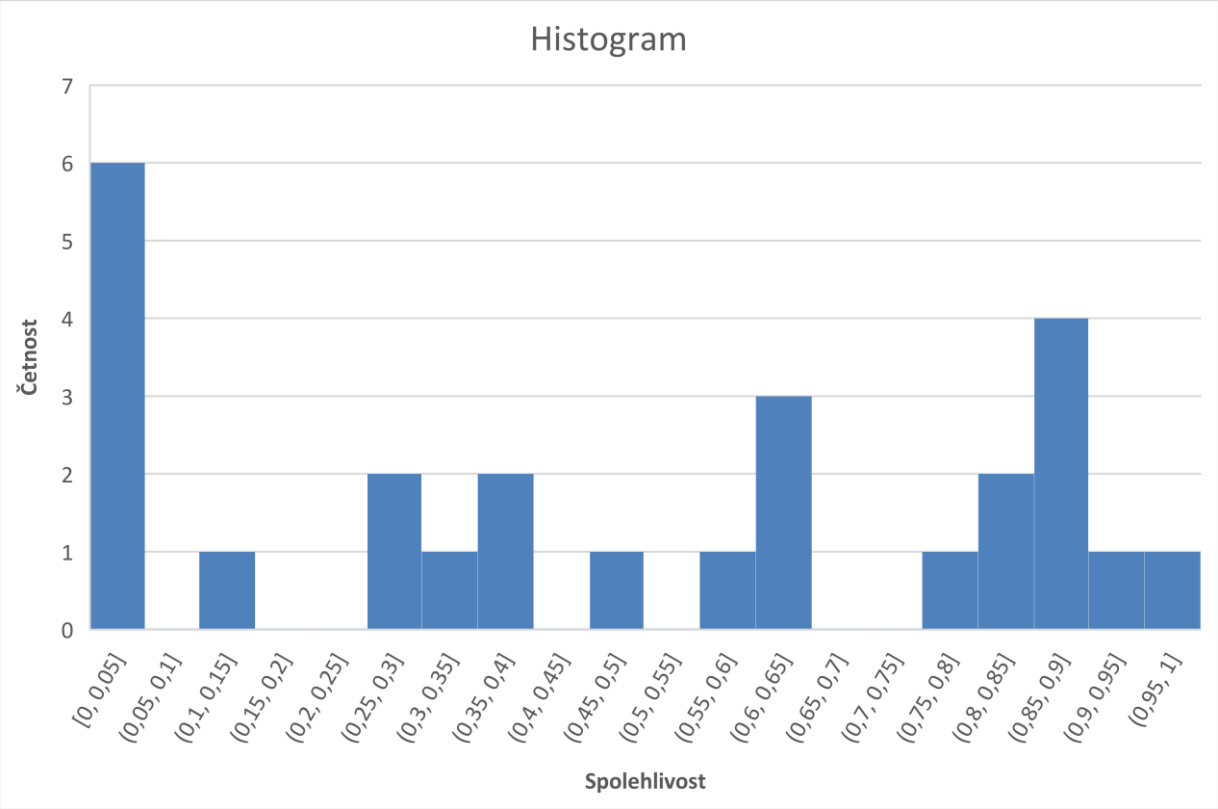
tříd jich 6 nezvládla síť rozpoznat vůbec (m203, m207, m208, m221, m227, m317), jak lze vidět jsou to učebny, které mají podobná čísla a síť pravděpodobně není dostatečně kvalitní na to, aby zvládla takto malé rozdíly rozpoznat. Dalších 8 tříd bylo rozpoznáno s nízkou spolehlivostí, jedná se o učebny m204, m210, m212, m219, m316, m340 a m341.

Po zanesení naměřených hodnot do tabulky vzniká Tabulka 12 s rozpisem spolehlivostí rozpoznání jednotlivých tříd.

Třída	**Spolehlivost**		**Třída**	**Spolehlivost**
M203	0		M227	0
M204	0,15		M229	0,86
M207	0		M230	0,96
M208	0		M231	0,65
M209	0,85		M238	0,76
M210	0,26		M313	0,89
M211	0,87		M315	0,62
M212	0,39		M316	0,32
M218	0,81		M317	0
M219	0,46		M339	0,56
M220	0,64		M340	0,4
M221	0		M341	0,26
M226	0,95		M343	0,86

Tabulka 12 Spolehlivost rozpoznání učebny

Po zanesení hodnot do histogramu spolehlivostí, který zobrazuje Obrázek 23, lze vidět, že po vynechání nulových hodnot, které značí štítky, které síť nezvládla rozpoznat se spolehlivost nejvíce pohybovala v rozmezí 0,85 – 0,9 a poté 0,6 – 0,65. Průměrná spolehlivost je 0,48, medián se nachází na hodnotě 0,51.



Obrázek 23 Histogram spolehlivosti

7 Závěry a doporučení

V práci byla provedena analýza některých frameworků určených pro rozšířenou realitu, zejména pak frameworků, které umí pracovat s rozpoznáváním grafických objektů v reálném prostředí. Z frameworků byl nakonec vybrán framework TensorFlow, který je univerzálně použitelný a má pro svou implementaci dostatečné množství dokumentačních zdrojů. Z frameworku TensorFlow bylo použito dvou funkcí, a to rozpoznávání obrázků a detekce objektů. Tyto funkce byly následně použity v mobilní aplikaci pro operační systém Android.

V průběhu implementace bylo testováno velké množství různých variant modelů a několik variant samotné aplikace pro rozpoznávání. Výsledkem pak je aplikace, která vstupní obraz analyzuje postupně dvěma sítěmi. Nejprve dojde k rozpoznání polohy štítku, poté je ze vstupního obrazu tato zájmová oblast oříznuta. K detekci informace, o jaký štítek se jedná, dochází již pouze na základě tohoto výřezu.

Bylo také zjištěno, že rozpoznávání objektů, které se od sebe liší pouze v malé části, je pro aktuální technologie velmi náročné. Navíc zde nastává problém s tím, že štítky ve škole se v průběhu školního roku mění, a samotný rozvrh hodin je na celém štítku významnější objekt než číslo místnosti.

Některé informace, které byly použity pro tvorbu této práce však v době čtení práce již nemusí být platné, a to zejména z důvodu velmi rychlého vývoje oblasti rozpoznávání objektů a rozšířené reality. Například testovaný model Inception pro klasifikaci štítku byl v této práci použit ve verzi 2, nyní již existuje verze 3. Ukázková mobilní aplikace pro operační systém Android, z které vychází použitá aplikace, také prochází postupným vývojem a je tedy možné, že se stejnými daty, ale novým modelem nebo aktualizací aplikace, mohou být při opakování testu výsledky jiné. S největší pravděpodobností lze očekávat, že výsledky by byly lepší, nelze to však prohlásit s jistotou.

Pro možnost navázání na tuto práci jsou na přiloženém CD poskytnuta trénovací data již rozdělená do tříd, také zde jsou k dispozici použité trénovací skripty a zdrojové kódy mobilní aplikace.

Existuje několik dalších možných postupů, které by mohly vést ke zlepšení detekce nebo rychlosti aplikace. První možností je natrénovat novou verzi modelu Inception se stejnými daty a poté provést otestování spolehlivosti. Druhou možnou cestou je pak natrénování aktuální MobileNet sítě pro rozpoznávání objektů na datech, která neobsahují štítek celý, ale pouze část, která obsahuje číslo místnosti. Možnou variantou je i vynechání rozpoznávání obrazových dat a číslo místnosti detekovat jinou metodou, například pomocí OCR.

Co se týká rychlosti aplikace tak je zde možností také více, je možné vyzkoušet funkcionalitu s YOLO detektorem objektů, který slibuje několikrát rychlejší detekci než při použití TensorFlow object detection API, otázkou zde zůstává, jestli tento detektor zvládne úspěšně určit polohu štítku. Jednou z možností zrychlení detekce je i varianta, kdy by byla vytvořena nějaká webová služba přijímající obrázky a vracející výsledek. Otázkou u použití tohoto cloudového řešení by bylo zejména vyřešení problému s datovou náročností a tím pádem nutnost provést analýzu vstupního obrazu, a v případě rozmazání snímku tento snímek ignorovat. Datovou náročnost by šlo také vyřešit tím, že by uživatel aplikace pro odeslání musel provést nějakou akci, například stisknout tlačítko. Je také otázkou, jestli by odesílání a přijímání dat přes počítačovou síť bylo rychlejší, než když detekce probíhá přímo v zařízení. Přínos takového řešení by určitě byl zejména při potřebě drobných změn modelu, kdy by stačilo změnit model, s kterým se pracuje na serveru, a aplikace by fungovaly dále.

Jednou z možností zvýšení přesnosti i rychlosti detekce pak zůstává možnost vytvořit celý model od začátku přímo na míru danému řešení. V takovém případě zde však nastává problém s náročností implementace takového řešení. Implementace takového řešení by byla natolik náročný proces, že pravděpodobně není možné na ní pracovat pouze v jedné osobě s klasickým hardwarem, který se vyskytuje v běžných domácnostech.

Cílem práce bylo provést analýzu existujících řešení pro rozpoznávání objektů v reálném prostředí a navrhnout možné řešení mobilní aplikace, která poté může být využita v aplikaci s rozšířenou realitou. Tento cíl byl splněn, nicméně bylo zjištěno, že vybrané řešení není plně vyhovující z důvodů nepřesnosti rozpoznávání, a u některých variant i z důvodu příliš dlouhé doby detekce a tím způsobenou

pomalostí aplikace. V předchozích odstavcích je popsáno několik možných postupů, kterými by bylo možné na tuto práci navázat pro zlepšení spolehlivosti rozpoznávání štítků.

8 Seznam použité literatury

1. **Niantic, Inc.** Pokémon Go Plus. [Online] 2016. [Citace: 2. 4 2018.] https://www.pokemongo.com/en-us/pokemon-go-plus/.

2. **HAIDER, Karrar.** 20 Best Augmented Reality Games For Smartphones (of 2017). *Hongkiat - Tech and Design Tips.* [Online] Hongkiat, 2018. [Citace: 2. 4 2018.] https://www.hongkiat.com/blog/best-ar-smartphone-games-2017/.

3. **FLÍDR, Jáchym.** Aplikace s rozšířenou realitou, které nesmíš minout. *#tech.* [Online] RED BULL, 2018. [Citace: 25. 5 2018.] https://www.redbull.com/cz-cs/tech-ar-aplikace-rozsirena-realita.

4. **KOOPER, Rob.** Virtual Reality. *Georgia Institute of Technology.* [Online] 17. 11 1995. [Citace: 23. 5 2018.] https://www.cc.gatech.edu/gvu/people/Masters/Rob.Kooper/Thesis/Chapter2.html.

5. **FURHT, Borko.** *Handbook of Augmented Reality.* Boca Raton : Springer, 2011. str. 3. ISBN 978-1-4614-0063-9.

6. **PTC Inc.** Vuforia | Augmented Reality. [Online] 2018. https://www.vuforia.com/.

7. **VisionStar Information Technology (Shanghai) Co., Ltd.** EasyAR-Best engine for developing Augmented Reality. [Online] 2012-2017. [Citace: 27. 12 2017.] https://www.easyar.com/.

8. **LAMB, Philip.** ARToolKit Home Page. *ARToolKit.* [Online] [Citace: 15. 3 2018.] https://www.hitl.washington.edu/artoolkit/.

9. **Google Inc.** *TensorFlow.* [Online] https://www.tensorflow.org/.

10. —. Graphs and Sessions. *TensorFlow.* [Online] 11. 7 2018. [Citace: 25. 7 2018.] https://www.tensorflow.org/guide/graphs.

11. **HE, Xingui a XU, Shaohua.** *Artificial Neural Networks.* Berlin, Heidelberg : Springer, 2009. ISBN 978-3-540-73761-2.

12. **TRENZ, Oldřich, FEJFAR, Jiří, POPELKA, Ondřej, KOLOMAZNÍK, Jan, ŠTENCL, Michael.** Neuronové sítě. [Online] Mendelova univerzita v Brně, 2010. [Citace: 25. 7 2018.] https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=21471.

13. **GOODFELLOW, Ian, BENGIO, Yoshua, COURVILLE, Aaron.** *Deep Learning (Adaptive Computation and Machine Learning).* Cambridge : The MIT Press, 2016. ISBN 978-0262035613.

14. **CORNELISSE, Daphne.** An intuitive guide to Convolutional Neural Networks. [Online] 24. 4 2018. [Citace: 26. 7 2018.] https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050.

15. **NG, Andrew, NGIAM, Jiquan, FOO, Chuan Yu, MAI, Yifan, SUEN, Caroline, COATES, Adam, MASS, Andrew, HANNUN, Awni, HUVAL, Brody, WANG, Tao, TANDON, Sameep.** Unsupervised Feature Learning and Deep Learning Tutorial. [Online] Stanford University, 2013. [Citace: 28. 7 2018.] http://ufldl.stanford.edu/tutorial/supervised/Pooling/.

16. **Google Inc.** How to retrain an Image Classifier for new Categories. *TensorFlow.* [Online] 25. 5 2018. [Citace: 22. 06 2018.] https://www.tensorflow.org/tutorials/image_retraining.

17. **CHU, Vincent.** We Need to Go Deeper: A Practical Guide to Tensorflow and Inception. *Medium.* [Online] 13. 2 2017. [Citace: 15. 7 2018.] https://medium.com/initialized-capital/we-need-to-go-deeper-a-practical-guide-to-tensorflow-and-inception-50e66281804f.

18. **SZEGEDY, Christian, LIU, Wei, JIA, Yangqing, SERMANET, Pierre, REED, Scott, a další, a další.** *Going Deeper with Convolutions.* 2017. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). arXiv:1409.4842.

19. **G. HOWARD, Andrew, ZHU, Menglong, CHEN, Bo, KALENICHENKO, Dmitry, WANG, Weijun, WEYAND, Tobias, ANDREETTO, Marco, HARTWIG, Adam.** *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* 17. 4 2017. arXiv:1704.04861.

20. **NVIDIA Corporation.** NVIDIA cuDNN. *NVIDIA Developer.* [Online] 2018. [Citace: 21. 1 2018.] https://developer.nvidia.com/cudnn.

21. **Python Software Foundation.** About Python. *Python.org.* [Online] 2001 - 2018. [Citace: 22. 6 2018.] https://www.python.org/about/.

22. **LUTZ, Mark, ASCHER, David.** *Naučte se Python.* [překl.] Jakub Mareček. Praha : Grada Publishing a.s., 2003. ISBN 80-247-0367-X.

23. **NVIDIA Corporation.** Cuda Toolkit. *NVIDIA Developer.* [Online] 2018. [Citace: 08. 07 2018.] https://developer.nvidia.com/cuda-toolkit.

24. **Google.** Tensors. *TensorFlow.* [Online] 2. 11 2017. [Citace: 21. 1 2018.] https://www.tensorflow.org/programmers_guide/tensors.

25. **Google Inc.** How to Retrain Inception's Final Layer for New Categories. *TensorFlow.* [Online] 27. 01 2018. [Citace: 28. 01 2018.] https://www.tensorflow.org/tutorials/image_retraining.

26. **EVERINGHAM, Mark, VAN GOOL, Luc, WILLIAMS, Chris, WINN , John, ZISSERMAN, Andrew.** The PASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision.* 2010, 88(2), stránky 303-338.

27. **Tzutalin.** LabelImg. *GitHub.* [Online] Git code, 2015. [Citace: 3. 4 2018.] https://github.com/tzutalin/labelImg.

28. **Stanford Vision Lab, Stanford University, Princeton University.** ImageNet. *ImageNet.* [Online] 2016. [Citace: 3. 4 2018.] http://www.image-net.org/.

29. **Google Inc.** Data IO (Python functions). *TensorFlow.* [Online] 2. 3 2018. [Citace: 3. 4 2018.] https://www.tensorflow.org/api_guides/python/python_io#tfrecords_format_details.

30. **KINSLEY, Harrison.** Creating TFRecords - Tensorflow Object Detection API Tutorial. [Online] PythonProgramming.net, 2017. [Citace: 3. 4 2018.] https://pythonprogramming.net/creating-tfrecord-files-tensorflow-object-detection-api-tutorial/.

31. **Google Inc.** A Tool Developer's Guide to Tensorflow Model Files. *TensorFlow.* [Online] 25. 5 2018. [Citace: 8. 7 2018.] https://www.tensorflow.org/extend/tool_developers/#freezing.

32. —. Building TensorFlow on Android. *TensorFlow.* [Online] 25. 5 2018. [Citace: 8. 7 2018.] https://www.tensorflow.org/mobile/android_build.

33. —. TensorFlow Android Camera Demo. [Online] Github, 2018. [Citace: 14. 7 2018.] https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android.

34. **REDMON, Joseph a FARHADI, Ali.** YOLO9000: Better, Faster, Stronger. *Joseph Redmon - Survival Strategies for the Robot Rebellion.* [Online] 2016. [Citace: 14. 7 2018.] https://pjreddie.com/darknet/yolov2/.

35. **HOLLEMANS, Matthijs.** Google's MobileNets on the iPhone. *Machine, Think!* [Online] 14. 6 2017. [Citace: 15. 7 2018.] http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/.

Obsah přiloženého CD

Přiložené CD obsahuje zdrojové kódy výsledné aplikace v adresáři *android_application,* dále obsahuje trénovací a další skripty použité v této práci (adresář *tensorflow_scripts)*.

V adresáři *training_data* jsou k dispozici učící data rozdělená do jednotlivých tříd, jsou zde k dispozici trénovací data jak pro metodu detekce polohy štítku, tak pro rozpoznávání obrazu. Data jsou roztříděna do podadresářů, jejichž název odpovídá tomu, co obsahují. V podadresáři object_detection jsou tedy trénovací data pro metodu detekce polohy štítku. Kdežto v adresářích training_set_XYZ jsou data pro rozpoznávání obrazu (XYZ je číslo, které znamená množství obrázků pro každou třídu).

Adresář *other_materials* obsahuje další materiály, které mohou být užitečné v případě pokračování práce. Nachází se tam aplikace labelimg v adresáři *label_img* a dále adresář *image_cropping*, který obsahuje autorem práce vytvořený skript *crop_images.py*, který dokáže na základě XML vygenerovaných aplikací labelimg, ořezat vstupní data do výřezů a tím usnadní tvorbu trénovacích dat založených pouze na výřezu štítku.

V adresáři *training_scripts* naleznete dva podadresáře – *image_detection* a *object_detection*. Adresáře obsahují potřebná trénovací data pro spuštění skriptu. U metody image detection je to jednoduché, zde stačí spustit skript *retrain.py* a zadat mu správné parametry. Parametry pro tento skript jsou popsané v práci. V adresáři *object_detection* se navíc nachází textový soubor *commands.txt*, který obsahuje užitečné příkazy, které se zadávají do příkazového řádku pro spuštění trénování object detection modelu, také se zde nachází soubor *help.txt*, který obsahuje stručný návod, jak spustit trénovací skript object_detection. Celý postup trénování modelu se poté nachází v textu práce v sekci implementace.

Jako poslední se na CD nachází adresář *thesis*, který obsahuje text práce v PDF a XLS soubor obsahující výsledky poslední testované sítě.

Oskenované zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Kombinovaná
Obor/komb.: Aplikovaná informatika (ai2-k)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Vlček Roman	Březová 53, Meziměstí - Březová	I1600278

TÉMA ČESKY:

Rozpoznávání objektů v obraze

TÉMA ANGLICKY:

Visual Object Recognition

VEDOUCÍ PRÁCE:

Ing. Pavel Kříž, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Provést rešerši existujících řešení pro implementaci rozpoznávání objektů a pro rozšířenou realitu (AR). Navrhnout vhodné řešení rozpoznávání objektů ("tagů") pro přípravu nasazení AR v mobilní hře určené pro sběr rádiových fingerprintů. Řešení rozpoznávání objektů implementovat na oper. systému Android. Otestovat spolehlivost rozpoznávání.

Osnova:
1. Úvod
2. Cíl
3. Rešerše dostupných knihoven a frameworků
4. Analýza a návrh řešení
5. Implementace
6. Testování a výsledky
7. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

1) https://thinkmobiles.com/blog/best-ar-sdk-review/
2) https://www.vuforia.com/
3) https://www.tensodrflow.org/
4) http://socialcompare.com/en/comparison/augmented-reality-sdks
5) Dhiraj Amin and Sharvari Govilkar: COMPARATIVE STUDY OF AUGMENTED REALITY SDK'S
6) https://arxiv.org/abs/1512.00567

Podpis studenta: ..... **Datum:** 10. 7. 18

Podpis vedoucího práce: ..... **Datum:** 10. 7. 18