VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VYHLEDÁVÁNÍ ZÁJMOVÝCH OBJEKTŮ VE VIDEU

OBJECT INSTANCE SEARCH IN VIDEO

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. BOHDAN IAKYMETS**
AUTHOR

VEDOUCÍ PRÁCE **Ing. BERAN VÍTĚZSLAV, PhD.**
SUPERVISOR

BRNO 2019

Abstrakt

Práce se zaměřuje na vytvoření mobilní aplikace, která bude pomáhat návštěvníkům galerií a muzeí snadněji obdržet informace o zajímavém objektu výtvarného umění. Aplikace zahrnuje práci s rozšířenou realitou. Mobilní aplikace slouží pro poskytovaní informace uživateli o zajímavém díle. Data pro všechny díla jsou ukládaný na straně serveru, který je také části projektu.

Abstract

This work focuses on creating mobile application, that helps visitors of galleries and museums to find, in a more easier way, interesting information about visual art objects.

Klíčová slova

Neuronové sítě, konvoluční neuronové sítě, hluboké učení, počítačové vidění, rozpoznávaní obrazů

Keywords

Artificial neural network, convolutional neural network, deep learning, computer vision, image recognition

Citace

IAKYMETS, Bohdan. *Vyhledávání zájmových objektů ve videu.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Beran Vítězslav, PhD.

Vyhledávání zájmových objektů ve videu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Vítězslava Berana. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Bohdan Iakymets
22. května 2019

Poděkování

Chtěl bych poděkovat všem kdo pomohl mi při práci nad diplomkou: vedoucímu pánovi Vítězslavu Beranovi a člověku který pomohl mi z češtinou Martinovi Betáku.

Obsah

Kapitola 1

Úvod

Umění vždy bylo důležitým faktorem pro lidskou civilizaci. Muzea a galerie jsou místa, kde každý člověk může najít nejen díla předchozích generací, ale také seznámit se s moderním uměním, které člověk často nechápe.

Cílem teto práce je vytvořit aplikaci, která pomůže lidem se seznámit s historií určitého díla v galerii pomocí vlastního smartphonu a rozšířené reality.

Práce je rozdělená na 5 částí (bez úvodu a závěru). V kapitole číslo 2 budou popsané pojmy a teorie, důležité pro pochopení dalšího návrhu a implementace. Třetí kapitola popisuje kompletní architekturu aplikace, včetně mockupu aplikace a diagramu databáze. Postup implementace a použité technologie jsou ve čtvrté kapitole. Poslední část je věnovaná testovaní aplikace.

Kapitola 2

Teorie

Táto kapitola se zabývá úvodem do terminologie a popisem základních technologií, které budou použité během návrhu aplikace.

2.1 Výtvarné umění

Výtvarné umění je umění vytvořené především pro vizuální vnímání. Skládá se různých druhů, jako například [4]:

- Malířství - aplikace laku, barvy nebo jiného prostředku na pevný povrch, obvykle štětcem.
- Sochařství - trojrozměrné umění vytvořené jedním ze čtyř základních procesů: řezba, modelování, lití, konstrukce.
- Grafika - vizuální prezentace na nějakém povrchu, jako je zeď, plátno, papír nebo kámen, která používá jako základní prostředky výrazy přímky, čáry, skvrny a tečky.
- Také do druhů výtvarného umění patří architektura, design a aplikované umění.

Kvůli tomu, že aplikace je zaměřená především na použiti v galerii nebo v muzeu, pro rozpoznávaní bude použito prvních tří druhů výtvarného umění.

2.2 Existující řešení

Existuje několik systémů, které implementuji instant image search, nejznámějším z nich je Google Images. Kvůli tomu, že Google Images je komerční produkt, není známo, jaké techniky algoritmus používá. Ale na základě výzkumu lze říct, že každý obrázek je označený klíčovými slovy a jeden ze způsobů ohodnocení podobností obrázků je na základě barvy.

2.3 Instant image search

Tradiční paradigma pro hledání obrázků se jmenuje Content-based image retrieval (CBIR). CBIR je nalezení obrázků na základě vizuálních vlastností, jako barva, struktura a tvar. Začala se vyvíjet ještě v 90. letech minulého století. V dnešní době je použitá pro hledaní podobných obrázků v Google Images, Pinterest, TinEye.

2.3.1 Hledaní podobného obrázků

Na hledaní v databázi podobných obrázků na základě vzorkového obrázku, CBIR používá dva kroky [9]:

1. Extrakce příznaků vzorkového obrázku.
2. Hledaní podobných příznaků v databázi (většinou příznaky jsou vektory, proto podobnost ohodnocuje podle vzdálenosti mezí vektory)

2.3.2 Příznaky

Neexistuje jediná přesná definice pojmu příznak, ale celkově lze pojem popsat jako část nebo vlastnost obrázku, která se používá v algoritmech počítačového viděni. Příznak se může skládat z několika příznaku. Různé algoritmy používají různé příznaky.[1]

Příznaky lze rozdělit na globální a lokální. Globální příznaky popisuji obecné vlastností obrázku, jako texturu, barvu a tvar. V případě lokálních příznaků obrázek je reprezentován na základě jeho lokálních struktur sadou lokálních deskriptorů příznaku extrahovaných z množiny obrazových oblastí nazývaných *oblasti zájmu* (tj. Klíčové body), jak je ilustrováno v 2.3. Většina místních vlastností představuje texturu v obraze obrazu.

Obrázek 2.1: Příklad extrakce lokálních příznaků

Deskriptor příznaků je algoritmus který vytvoří vektory/deskriptory příznaků.

2.3.3 Extrakce příznaků

První algoritmus pro extrakci lokálních příznaků je Scale-invariant feature transform (SIFT). Varianty algoritmu se dosud používají. Hlavní myšlenka SIFT je najít zajímavé body a pak vlastností lokálního regionu kolem bodu. Takový druh algoritmů se taký jmenuje "Handcrafted Features", což znamená extrahovaní příznaků jenom podle informace z obrázku. [8]

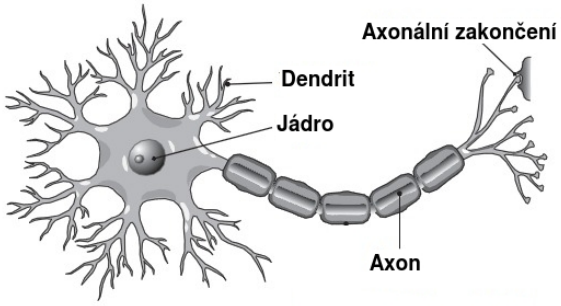
Ale od roku 2013 se vyvíjí jiný typ - "Learning-based Feature". Oproti předchozímu typu je Learning-based založen na předtrénované Konvoluční neuronové sitě (Convolutional neural network/CNN). Hlavní výhoda je menší nárok na paměť. Jako deskriptory Learning-based používá aktivace jednotlivých vrstev CNN.

2.4 Neuronové sítě

Neuronová síť je výpočetní nebo logická schéma postavená z homogenních prvků procesoru, což jsou zjednodušené funkční modely neuronů.[13]

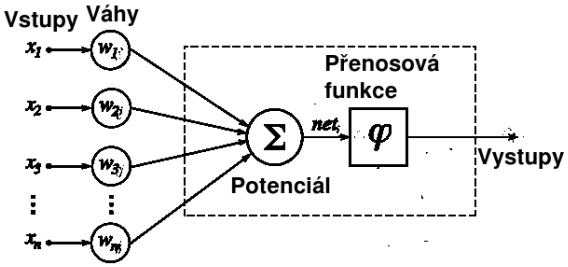
2.4.1 Neurony

Umělé neuronové sítě jsou založeny na biologických modelech, které tvoří dohromady síť nervových buněk. Neuronové sítě se skládají z neuronu, které komunikuji mezi sebou. Neuron začne přenášet na axony signál, když buzení neuronu přesahuje určitou prahovou hodnotu. Synapse - spojující uzly mezi axony jednoho neuronu a dendrity druhého, s jejich pomocí výstupní signál jednoho neuronu je přenášen na vstup dalšího. Synapse může jak posílit, tak oslabit signál. Obrázek 2.2 ukazuje strukturu biologického neuronu.



Obrázek 2.2: Biologický neuron

Jak již bylo zmíněno dříve, umělé neuronové sítě jsou zjednodušené modely biologických neuronových sítí. Umělý neuron přijímá jako vstup vektor hodnot, které jsou tedy násobení příslušnou vahou, pak tato hodnota je v těle neuronu sečtena a od výsledné hodnoty je odečten práh. Následkem této operace je vstup do přenosové funkce, který se ještě jmenuje potenciál neuronu, jehož výsledkem je výstupní hodnota neuronu. Obrázek 2.3 ilustruje strukturu umělého neuronu.



Obrázek 2.3: Model neuronu

2.4.2 Potenciál neuronu

Potenciál neuronu je součet vstupních signálů vynásobený jejich příslušnými váhami minus práh:

$$net = \sum_{i=1}^{N} w_i x_i - \theta \tag{2.1}$$

kde x_i je i-tý vstupní signál, w_i je i-tá synaptická váha, N je počet vstupních signálů a θ je práh neuronu.

Úlohou potenciálu je shromáždění všech vstupních signálů do jediného čísla - váženého součtu, který charakterizuje přijatý signál.

2.4.3 Přenosová funkce

Přenosová funkce je funkce, která dostává potenciál neuronu jako argument. Hodnotou takové funkce je výstup neuronu. Použití přenosové funkce umožňuje zpracovat komplexní nelineární závislosti neuronových sítí:

$$out = \varphi(net) \tag{2.2}$$

Nejběžnější přenosové funkce jsou:

- Skoková přenosová funkce. Tato funkce implementuje následující pravidlo: pokud je vstupní hodnota menší než prahová hodnota, hodnota aktivační funkce je nula, jinak hodnota funkce je jedna:

$$\varphi(net) = \begin{cases} 1 & \text{pro } net > \theta \\ 0 & \text{pro } net < \theta \end{cases} \tag{2.3}$$

- Sigmoidální přenosová funkce je funkce, která je diferencovatelná v celém svém definičním oboru. Tak malé změny argumentů funkcí vedou k malým změnám ve funkčních hodnotách:

$$\varphi(net) = \frac{1}{1 + e^{-net}} \tag{2.4}$$

  Obor hodnot takové funkce spočívá v intervalu $\langle 0, 1 \rangle$. Tato funkce je snadno diferencovatelná, což ji umožňuje používat při výuce neuronových sítí metodou zpětného šíření chyby.

- Přenosová funkce hyperbolické tangenty je bipolární aktivační funkce, která je diferencovatelná v celém svém definičním oboru:

$$\varphi(net) = \frac{2}{1 + e^{-knet}} - 1 \tag{2.5}$$

  Obor hodnot takové funkce spočívá v intervalu $\langle -1, 1 \rangle$. Tato aktivační funkce umožňuje zpracovávat záporné hodnoty.

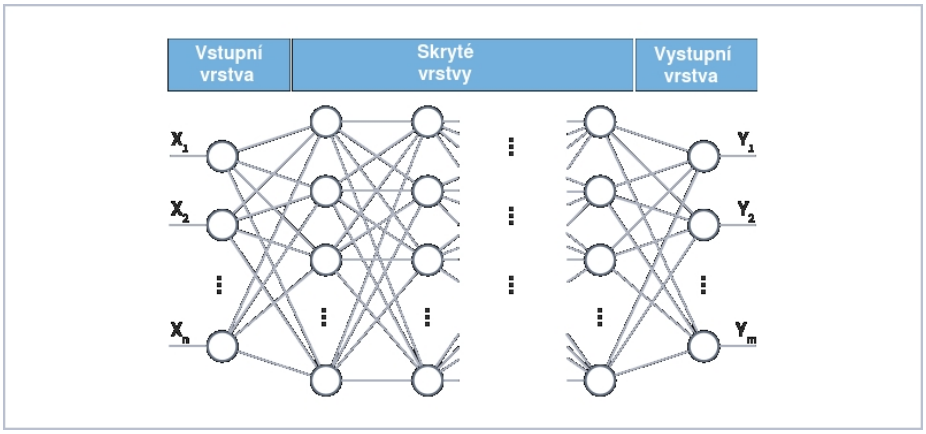
- ReLU (rectified linear unit) provede následující operaci: přeskočí pouze ty hodnoty argumentu, které jsou větší než nula.[10] Tak dojde k tomu, že výpočet hodnot samotné funkce i jejího derivátu proběhne mnohem rychleji než v sigmoidální funkce a funkce hyperbolické tangenty. Proto použití ReLU může významně zvýšit rychlost učení:

$$\varphi(net) = max(0, net) \tag{2.6}$$

2.4.4 Vrstvy

Vrstva, na kterou neurony obdrží vektor vstupních hodnot, se jmenuje vstupní vrstva. Její neurony prostě rozdělí vstupní signál mezi neurony další vrstvy. Poslední vrstva se jmenuje výstupní. Výstup této vrstvy je výsledkem práce neuronové sítě. Mezi vstupní a výstupní vrstvami mohou být umístěny skryté, nebo vnitřní vrstvy, ve kterých probíhá základní

zpracování dat. Sítě se skrytými vrstvami se nazývají vícevrstvá. Počet neuronů a počet skrytých vrstev se určuje podle typu řešeného problému. Na obrázku 2.4 lze vidět typickou vícevrstvou neuronovou síť.



Obrázek 2.4: Příklad vícevrstvé neuronové sítí

2.5 Konvoluční neuronové sítí

Konvoluční neuronové sítě (CNN) jsou třída hlubokých neuronových sítí a jsou velmi podobné běžným: jsou také postaveny na základě neuronů, které mají různé váhy a odchylky. Každý neuron přijímá některá vstupní data, provádí skalární součin informací a v určitých situacích doprovází tuto nelinearitu, ale oproti běžným sítím je založena na konvoluci a nejsou všechny neurony propojeny mezí sebou, každý z neuronů je spojen pouze s malou oblasti v obraze, což dovolí efektivně zpracovávat obrázky.

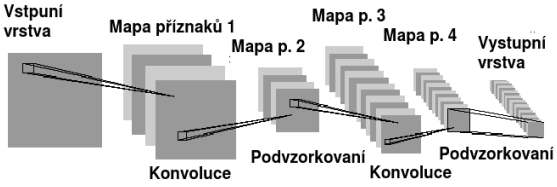
Konvoluce je operace, která ukazuje „podobnost“ jedné funkce s odraženou a posunutou kopií druhé. [11]

Vlastnosti zpracování dat v konvolučních neuronových sítích jsou následující:

- Lokální vnímání - na vstup jednoho neuronu je přiváděn ne celý obraz (výsledek předchozí vrstvy), ale pouze jeho část. Což umožňuje zachovat topologii obrazu od vrstvy k vrstvě.
- Sdílené váhy jsou malé množství váhů pro velké množství spoje.
- Snížení rozměru - prostorový rozměr obrazu se postupně snižuje kvůli podvzorkovacím vrstvám

Rozpoznávání objektů se provádí bez ohledu na velikost. V tomto případě přítomnost příznaku je víc důležitější než místo jeho přesného umístění na obrázku.

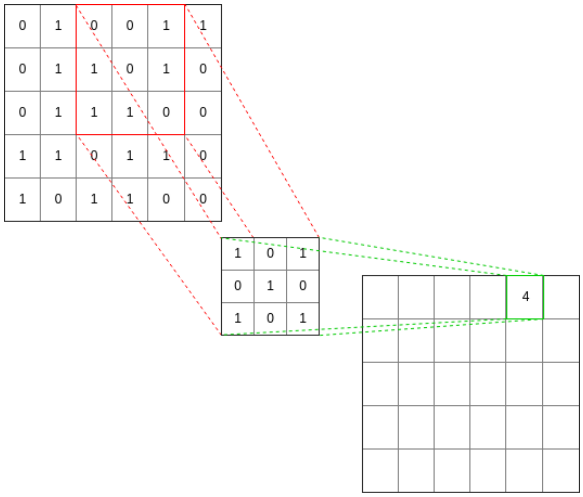
Konvoluční neuronová síť se skládá z po sobě jdoucích vrstev konvoluce a vrstev podvzorkování. Což umožňuje vytvářet mapy příznaků na aktuální vrstvě na základě zpracování map příznaků v předchozí vrstvě. Takovým způsobem se provádí hluboká analýza složitých hierarchií různých příznaků, výběr a rozpoznání místních příznaků. Na obrázku 2.5 lze vidět typickou architekturu konvolučních neuronových sítí.



Obrázek 2.5: Příklad konvoluční neuronové sítí

2.5.1 Konvoluční vrstva

V konvoluční vrstvě obraz je zpracován pomocí konvolučních filtrů, to znamená, že na každý obrazový bod je aplikován filtr, jako zobrazeno na obrázku 2.6.



Obrázek 2.6: Znázorněni konvoluce obrázku

Typický vzorec konvoluce:

$$(f * g)(x, y) = \sum_{k,l} (f(m - k, n - l)g(k, l)) \tag{2.7}$$

kde f - vstupní matice obrázku; g - jádro filtru; k - číslo řádku jádra; l - číslo sloupce jádra.

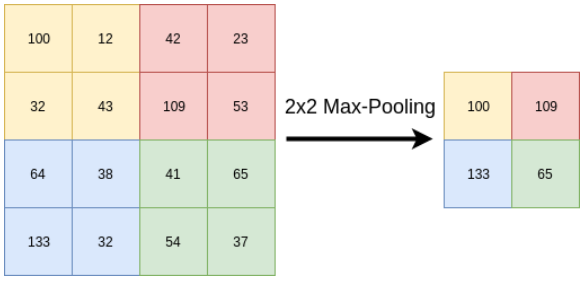
Jádro filtru je množina váhových koeficientu. Výsledkem této operace je také obrázek, který se jmenuje mapa příznaku. V závislosti na vybraném jádře filtru na mapě bude zvýrazněna nějaká určitá vlastnost vstupního obrazu. Pro nejkomplexnější výběr vlastností vstupního obrazu se používá několik různých konvolučních jader, na výstupu konvoluční vrstvy v tomto případě bude několik map příznaků.

Koeficienty jádra záleží na datech na kterých byla síť vycvičena. Tak například, pokud byla síť vycvičena na množství obličejů, pak nějaké jedno jádro by mohlo během učení vracet největší signál v oblasti očí, ústy, obočí nebo nosu, jiné jádro by mohlo vyhledávat jiné příznaky. Velikost jádra je zvolena tak, aby velikost map konvoluční vrstvy byla rovnoměrná, což umožňuje neztrácet informace při snižování rozměru ve vrstvě podvzorkovaní.

Výsledek skalární konvoluce pak jde na vstup přenosové funkce. Dřív pro tento účel tradičně používali funkce hyperbolické tangenty nebo sigmoidální. V dnešní době nejčastěji používaná je funkce ReLU, která umožňuje zrychlit proces učení a zjednodušit výpočty díky své jednoduchosti.

2.5.2 Podvzorkovací vrstva

Podvzorkovací vrstva (nebo pooling vrstva) redukuje velikost map příznaků, což redukuje velikost předávaných do sítí dat. Takové transformace jsou prováděny na nepřetržitých obdélnících (maska), každý z nich je pak stlačený na velikost 1x1. Nejpoužívanější je tato vrstva při použití masky 2x2 aplikované s krokem 2. Nejčastěji se při výběru příznaků používá funkce max-pooling, která vypočítává maximální hodnotu na zpracovávané mapě. Na obrázku 2.7 znázorněny příklad podvzorkovaní.



Obrázek 2.7: Příklad max-pooling podvzorkovaní

Kromě max-pooling funkce v podvzorkovací vrstvě mohou být použité jiné funkce, například funkce průměru (average-pooling) nebo funkce normalizace L2, ale v praxi se maximální funkce ukázala jako nejefektivnější.

2.5.3 Plně propojená vrstva

Během procesu procházeni obrázkem vrstev konvoluce a podvzorkování počet kanálů se zvětšuje, ale velikost obrazu se snižuje. Nakonec po průchodu několika vrstvami se mapa příznaků redukuje na vektor, ale počet map se významně zvýší. Vektory jsou deskriptory příznaků původního obrazu. Je pak sjednotí a dá na vstup běžné plně propojené sítě, která dokončí analýzu obrazu.

2.6 Přehled CNN architektur

V roce 2013 pomoci CNN v sadě dat ImageNet byla překonána bariéra 15% chyb klasifikace tisíc objektových typů. Od té doby bylo po dobu 5 let navrženo a vyškoleno mnoho různých modelů neuronových sítí a byla překonána meta s 5% chybami. Nejúspěšnější z nich jsou: VGG16, ResNet50, Inception, MobileNet, a mnoho dalších. Většina z nich je postavena na základě konvolučních neuronových sítí.

2.6.1 Inception modul

Obvykle pro zvětšení přesnosti CNN je nutně zvětšit samotný model (zvětšit šířku, zvětšit hloubku). Ale v tomto případě síť bude potřebovat víc zdrojů počítače. Proto jako řešení problému bylo navrhnuté použit Inception-modul.

Hlavní myšlenka modulu Inception je místo toho aby vybrali nějakou určitou velikost konvolučního jádra, použit několik možností najednou, a výsledky zřetězit. To však významně zvyšuje počet operací, které je třeba provést pro výpočet aktivace jedné vrstvy, ale kvůli jednomu triku lze redukovat tento počet: provést konvoluci s jádrem 1x1 před každým konvolučním blokem, tato operace se jmenuje Pointwise Convolution, což sníží rozměr

Model	Velikost obrázku	Velikost vah	Top-1 přesnost	Top-5 přesnost	Parametry	Hloubka
Xception	299 x 299	88 MB	0.790	0.945	22,910,480	126
InceptionV3	299 x 299	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	299 x 299	215 MB	0.803	0.953	55,873,736	572
MobileNet	224 x 224	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	224 x 224	14 MB	0.713	0.901	3,538,984	88

Tabulka 2.1: Porovnaní architektur používajících Inception modul

Model	Top-1 Přesnost	Průměr procentu předpokladu
Xception	100%	58.458%
InceptionV3	94.1%	87.039%
InceptionResNetV2	87.5%	83.576%
MobileNet	95.83%	94.423%
MobileNetV2	95.83%	93.71%

Tabulka 2.2: Porovnaní přesnosti architektur na uměleckých dílech

signálu, který dojde na vstup konvolucí s velkými velikostmi jádra. Což dovolí relativně zmenšit nárok na zdroje počítače při zvětšení výkonu.[2]

Dnes se v modelech čistý Inception modul nepoužívá, místo něho jsou architektury založeny na Depthwise Separable Convolutions, který má následující minimální odlišnosti oproti extrémní verzi Inception:

- Pořadí operací: Depthwise Separable Convolution je obvykle implementováno tak, že jako první provede prostorovou konvoluci kanálů a potom konvoluci 1x1, což Inception modul děla v opačném pořadí.
- Přítomnost nebo nepřítomnost nelinearity po první operace. Depthwise Separable Convolutions je implementovano bez nelinearity, kdy Inception má po obojích operacích nelinearitu ReLU.

Modul Inception a Depthwise Separable Convolution úspěšně používají takové už zmíněné dřív modely: MobileNet, MobileNetV2, InceptionResNetV2, InceptionV3 a také Xception.

Další tabulka ukazuje výsledky testovaní na datech které obsahuji umělecké dílo. Na učení pro každé dílo byly použity 3 obrázky. Na testovacích obrázcích každé dílo mělo jiné pozadí aby se redukoval vliv prostředí na výsledek. 2.2

V tabulce je použitá hodnota Průměr procentu předpokladu, kterou lze vyjádřit jako $ppp = \frac{\sum \text{Správné odpovědí}}{n}$, kde n - je počet správných odpovědí. Hlavní myšlenka průměru je ukázat která z architektur má lepší výběr příznaků a při odlišném pozadí testovací vektor příznaků má větší shodu s originálním vektorem. Jako lze vidět MobileNet architektura není 100% přesná, ale oproti ostatním má největší průměr předpokladu, tzn. že průměr podobností správných testovacích obrázků je 94.423% což je víc než ostatní. Proto během extrakce příznaků aplikace bude používat architekturu MobileNet.

2.6.2 MobileNet

MobileNet (zkratka MobileNetwork, mobilní síť)[6] je třídou efektivních modelů neuronových sítí navržených pro použití v mobilních a vestavěných aplikacích pro počítačové vidění. Tato architektura je založena na použití Depthwise Separable Convolution. Na rozdíl od mnoha jiných modelů zaměřených na stejné třídy zařízení, primárním cílem jejich vývoje bylo snížit velikost neuronové sítě, zatímco primárním cílem při vývoji MobileNet bylo zkrátit časové zpoždění, ke kterému dochází během provozu sítě, a zmenšení velikosti je pouze sekundární cíl.

Kapitola 3

Použité technologie

Tato kapitola popisuje technologie použité během vývoje.

3.1 Existující knihovny pro prací s neuronovými síti

Pro implementaci algoritmů pro práci s neuronovými sítěmi v rozvinutém systému bylo rozhodnuto použít jednu ze stávajících knihoven. Proto byla provedena analýza stávajících softwarových řešení pro implementaci algoritmů hlubokého učení a na základě výsledků této analýzy byl proveden výběr.

Pro srovnaní byly uvažovány takové knihovny jako Deeplearning4j, Theano, Pylearn2, Torch, Caffe a Keras. Podrobnější popis těchto knihoven:

- Deeplearning4j je open source knihovna pro implementaci neuronových sítí a algoritmů hlubokého učení, napsaných v jazyce Java. Lze jej použít z jazyků Java, Scala a Clojure, je podporována integrace s Hadoop, Apache Spark, Akka a AWS. Knihovna je vyvíjena a spravována společností Skymind, která také poskytuje komerční podporu této knihovně. Uvnitř této knihovny se používá knihovna pro rychlou práci s n-rozměrnými poli ND4J vyvinutými stejnou společností. Deeplearning4j podporuje mnoho typů sítí, včetně vícevrstvého perceptronu, konvolučních sítí, omezených Boltzmanových strojů, skládaných denoizačních automatických kodérů, hlubokých automatických kodérů, rekurzivních autoenkodérů, sítí s hlubokým přesvědčením, opakujících se sítí a dalších. Důležitým rysem této knihovny je její schopnost pracovat v clusteru. Knihovna také podporuje učení sítě pomocí GPU.

- Torch je knihovna pro vypočítaní a implementace algoritmů strojového učení (machine learning) implementovaná v jazyce C, ale umožňuje výzkumným pracovníkům používat s ní mnohem pohodlnější skriptovací jazyk Lua. Tato knihovna poskytuje vlastní efektivní implementaci operací na maticích, vícerozměrných polích, podporuje výpočty na GPU. Umožňuje implementovat plně propojené a konvoluční sítě. Má otevřený zdrojový kód.

- Theano je otevřená knihovna v jazyce Python, která umožňuje efektivně vytvářet, počítat a optimalizovat matematické výrazy pomocí vícerozměrných polí. Knihovna používá NumPy k reprezentaci vícerozměrných polí a provedení operací. Tato knihovna je určena především pro vědecký výzkum a byla vytvořena skupinou vědců z University of Montreal. Schopnosti Theana jsou velmi široké a práce s neuronovými sítěmi je

pouze jednou z jeho malých částí. Tato knihovna je zároveň nejoblíbenější a nejčastěji se zmiňuje, pokud jde o práci s hlubokým učením.

- Pylearn2 je open source python knihovna postavená na Theanu, která však poskytuje pohodlnější a jednodušší rozhraní pro výzkumné pracovníky, poskytuje připravený soubor algoritmů a umožňuje jednoduchou konfiguraci sítí pomoci souborů YAML. Vyvinutý skupinou vědců z laboratoře LISA na univerzitě v Montrealu.

- Caffe je knihovna zaměřená na efektivní implementaci algoritmů hlubokého učení (deep learning), vyvinutých především Berkley Vision and Learning Center, ale stejně jako všechny předchozí, má otevřený zdrojový kód. Knihovna je implementována v jazyce C, ale také poskytuje pohodlné rozhraní pro Python a Matlab. Podporuje plně propojené a konvoluční sítě, umožňuje popsat sítě jako sadu vrstev ve formátu .prototxt, podporuje výpočty na GPU. Mezi výhody knihovny patří také přítomnost velkého množství předem naučených modelů a příkladů, které v kombinaci s ostatními charakteristikami činí z knihovny tu nejjednodušší pro práci mezi výše uvedenými.

- Keras je open source knihovna Keras pro práci s neuronovými sítí napsaná v Pythonu. Je nadstavbou nad TensorFlow, CNTK, a Theano. Je určena pro rychlou a snadnou práci se sítěmi hlubokého učení. Zaměřuje se na minimalitu, modularitu a rozšiřitelnost. Byla vytvořena jako součást výzkumného projektu ONEIROS (Open Neuro-Electronic Intelligent Robot Operating System), jehož hlavním autorem a vývojářem je inženýr společnosti Google - François Cholet. Knihovna má předem vycvičené modely a podporuje učení sítě pomocí GPU.

Podle souboru kritérií byly pro další experimenty vybrány 2 knihovny: Deeplearning4j a Keras. Tyto 2 knihovny byly instalovány a testovány v praxi.

Mezi těmito knihovnami se Deeplearning4j ukázal jako nejproblematičtější v instalaci (nejvíc problémů se objevilo při instalaci na Android), navíc byly odhaleny chyby v ukázkových příkladech dodávaných s knihovnou, což způsobilo několik otázek o spolehlivosti knihovny a bylo velmi obtížné ji dále studovat. S ohledem na nižší produktivitu jazyka Java v srovnaní s Python, na kterém je implementovan Keras, bylo rozhodnuto zavrhnout další použití této knihovny.

Knihovna Keras se ukázala jako snadno instalovatelná a konfigurovatelná, a největší výhoda je v tom, že existuje celá řada kvalitních a dobře strukturovaných dokumentů a příkladů pracovního kódu pro tuto knihovnu, takže nakonec bylo možné přizpůsobit práci knihovny včetně použití grafické karty. Proto bylo rozhodnuto použit pro implementace knihovnu Keras.

3.2 Existující mobilní operační systémy

V současné době existuje mnoho různých mobilních operačních systémů. Ale všichni mají odlišnou popularitu. Pokud aplikace není multiplatformní, pak vývoj pro záměrně úzký okruh uživatelů ji může předem odsoudit k selhání. Čím více uživatelů bude aplikace vyzkoušet, tím větší bude zájem. Jedním z hlavních úkolů je proto vybrat operační systém, pro který bude vytvořena aplikace. Podle průzkumu IDC (International Data Corporation)[7] a osobních zkušeností byly vybrané následující operační systémy, které je třeba zvážit (v pořadí rostoucí popularity):

- iOS (14,1 %)

- Android (85,9 %)

3.2.1 iOS

iOS je operační systém pro smartphony, elektronické tablety a nositelné přehrávače, vyvinutý americkou společností Apple. Podíl iOS na trhu mobilních operačních systémů (14,1%) přesně odráží tržní podíl telefonů iPhone na trhu smartphonů, protože Systém iOS je nainstalován pouze na telefonech iPhone a iPad (mobilní zařízení Apple), zatímco Android je použitý v mobilech od rožních výrobců. Podíl iOS na trhu v tomto roce vzrostl: v prvním čtvrtletí roku 2017 to bylo 14,7% a v roku 2018 ve stejném čtvrtletí bylo 15,7%. Počet aplikací iOS v aplikaci App Store přesahuje dva miliony.

Tento operační systém je velmi atraktivní pro rozvoj, protože má široké publikum uživatelů. Přesto však existuje celá řada nevýhod:

- počet uživatelů iOS je stále menší než počet uživatelů Androidu;
- drahý vývojářský účet ke nahrávaní aplikace v App Store;
- nedostatek potřebných nástrojů pro vývoj bez počítače od Apple.

3.2.2 Android

Android je operační systém pro smartphony a mnoho dalších zařízení. Je založen na přizpůsobeném jádru Linuxu a virtuálním počítači Dalvik [8]. Původně vyvinutý společností Kalifornie založenou společností Android Inc., kterou pak koupil americký vyhledávací gigant Google. Podíl Androidu na trhu OS je 85,9%, což je mírný pokles oproti stejnému čtvrtletí 2017 roku 87,6%. Počet aplikací pro Android v úložišti aplikací Google Play přesahuje 2,6 milionu, tento operační systém má největší počet uživatelů, což přibere více zpětné vazby od uživatelů, kteří aplikaci nainstalovali. Účet vývojáře pro publikování aplikace do obchodu Play je také placen, ale stojí mnohem méně než v App Store a platba je provedena jednou (v App Store roční platba).

3.2.3 Základní struktura Android aplikaci

Vývoj Android aplikace primárně probíhá v programovacím jazyce Java, ale kromě toho lze použit jakýkoliv jazyk, který pak lze přeložit v JVM bytecode, a potom převést na Dalvik bytecode. To se děje tak pro optimalizaci běhového prostředí pro mobilní zařízení. Využití formátu .class jazyka Java má další výhodu, že lze využit většinou stávajících knihoven a nástrojů v jazyce Java při vývoji mobilní aplikace. Android aplikace se obvykle skládá z několika komponent [4]. Tyto komponenty jsou Aktivity, Služby, Vysílací přijímače a Obsah Poskytovatelé. Všechny komponenty, které aplikace používá, jsou uvedeny v AndroidManifest.xml. Tento soubor obsahuje také různá metadata aplikace, jako jsou požadovaná oprávnění, exponované služby, obsah filtrů a dalších informací. Každá aplikace pro Android musí tento soubor poskytnout.

Android aplikace používá okna (podobně jako v Windows), nicméně v tomto systému mají výše uvedená okna jiný název - `Activity`. Stejně jako ve Windows každé okno má vlastní životní cyklus a vlastní metody:

- Metoda `onCreate()` je volána při vytváření nového okna, během vývoje je tato metoda předefinována a aplikace a její komponenty jsou v ní inicializovány.

- Metoda `onStart()` je volána před zobrazením okna, nebo když aplikace je otevřena po předchozím zastavení.
- Metoda `onResume()` funguje podobně jako `onStart()`, ale z tím rozdílem, že taký se vola v případě když aplikace z pozadí přechází na popředí.
- Metoda `onPause()` se volá když aplikace přechází na pozadí, ale dosud může být viditelná pokud uživatel používá multi-window mode.
- Metoda `onStop()` se volá když aplikace není viditelná pro uživatele, nebo když aplikace je dokončená.
- Metoda `onDestroy()` je volána když uživatel dokončí aplikaci, v této metodě je možně uložit uživatelská data a parametry.

3.3 Existující webové frameworky

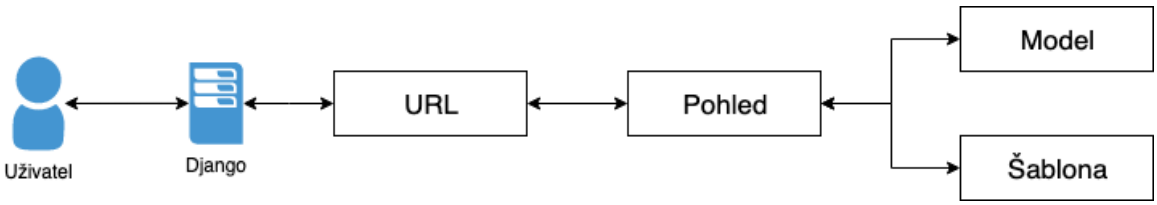
Kvůli tomu, že vývoj bude probíhat pomoci knihovny Keras, která je napsaná v jazyce Python a taký kvůli tomu, že Python je velice populárni věděcký jazyk programovaní, proto pro jazyk serveru bylo rozhodnuto použit Python.

Srovnaní bude zaměřené na nejpopulárnější webové frameworky: Django, Flask a Pyramid. Podrobnější popis těchto frameworků:

- Django je nejpopulárnější open source webový framework v jazyce Python, obsahuje spoustu vestavěných funkci, jako například vlastní ORM (Objektově relační mapování), který podporuje takové databáze jako PostgreSQL, MySQL, SQLite, a Oracle, nebo vlastní šablonovací systém. Jedna z největších výhod je obrovská komunita a spousta materiálů a dokumentace. Kromě toho framework zajišťuje bezpečnost a škalovatelnost pro velké a náročné projekty. V dnešní době Django použivaji takové gigantovi jako Public Broadcasting Service, Instagram, Mozilla, The Washington Times, Bitbucket a ostatní.
- Flask je open source microframework, druhý nejpopulárnější framework, má 12 000 forků a 44 000 hvězd na githubu. Hlavní myšlenka Flasku je udělat malý framework, snadně rozšiřitelný o potřebné funkce. Oproti Django nemá vestavěný ORM, ale je možnost zapojit libovolný podle potřeby. Má svůj šablonovací systém podobný jako ve Django. Kromě toho je kompatibilní s Google App Engine. Ve dnešní době je použitý takovymi firmami jako Pinterest a LinkedIn.
- Pyramid je open source Python webový framework, hlavně zaměřený na jednoduchost a rychlost vývoje. Hlavní výhoda Pyramid je to framework stejně výborně funguje s malými a velkými aplikacemi. Ale oproti Django nemá vestavěný ORM a šablonovací systém. Pyramid je celkem malý framework podobně Flasku pro rychlý a snadný vývoj, všechny chybějící funkce lze přidat pomoci add-onů.

Během experimentů vybraných frameworků pro další použití byl vybrán Django. Kvůli tomu, že Django má vestavěnou funkcionalitu kterou nemají ani Flask, ani Pyramid, jako Django ORM a webové rozhraní pro administrátora. Taky výběru přispěl fakt, že autor práce má praxi v použití Django.

3.3.1 Django



Obrázek 3.2: Architektura Django

Architektura Django je založená na modelu MTC (Model-Template-Controller)[5], ji lze vidět na obrázku 3.2, kterou lze rozdělit na tři častí:

- Model (model) je softwarový modul v rámci aplikace, která je jakýmsi prostředníkem mezi databází a moduly projektu. Model plní následující funkce:
  - Popisuje strukturu databáze v terminologii použitého programovacího jazyka;
  - Představuje data čtená z databáze v terminologii použitého programovacího jazyka;
  - Implementuje mechanismy vzorkování, třídění a filtrování dat obsažených v databázi;
  - Implementuje mechanismy pro editaci, mazání a přidávání dat do databáze;
  - Zkontroluje správnost zadaných dat (typová shoda, délka řetězce atd.)
- Template (šablona) je část, která určuje, jaká data musí být přijata a jak ji zobrazit, je zpracována pohledy a šablonami.
- Controller (řadič) je část, která vybere zobrazení založené v zavislostí od uživatelského vstupu. Controller je zpracovaný samotným vývojovým prostředím podle schématu URL, který nakonfigurovaný programátorem, a zavolá příslušnou funkci Pythonu pro zadanou adresu URL.

3.4 OpenCV

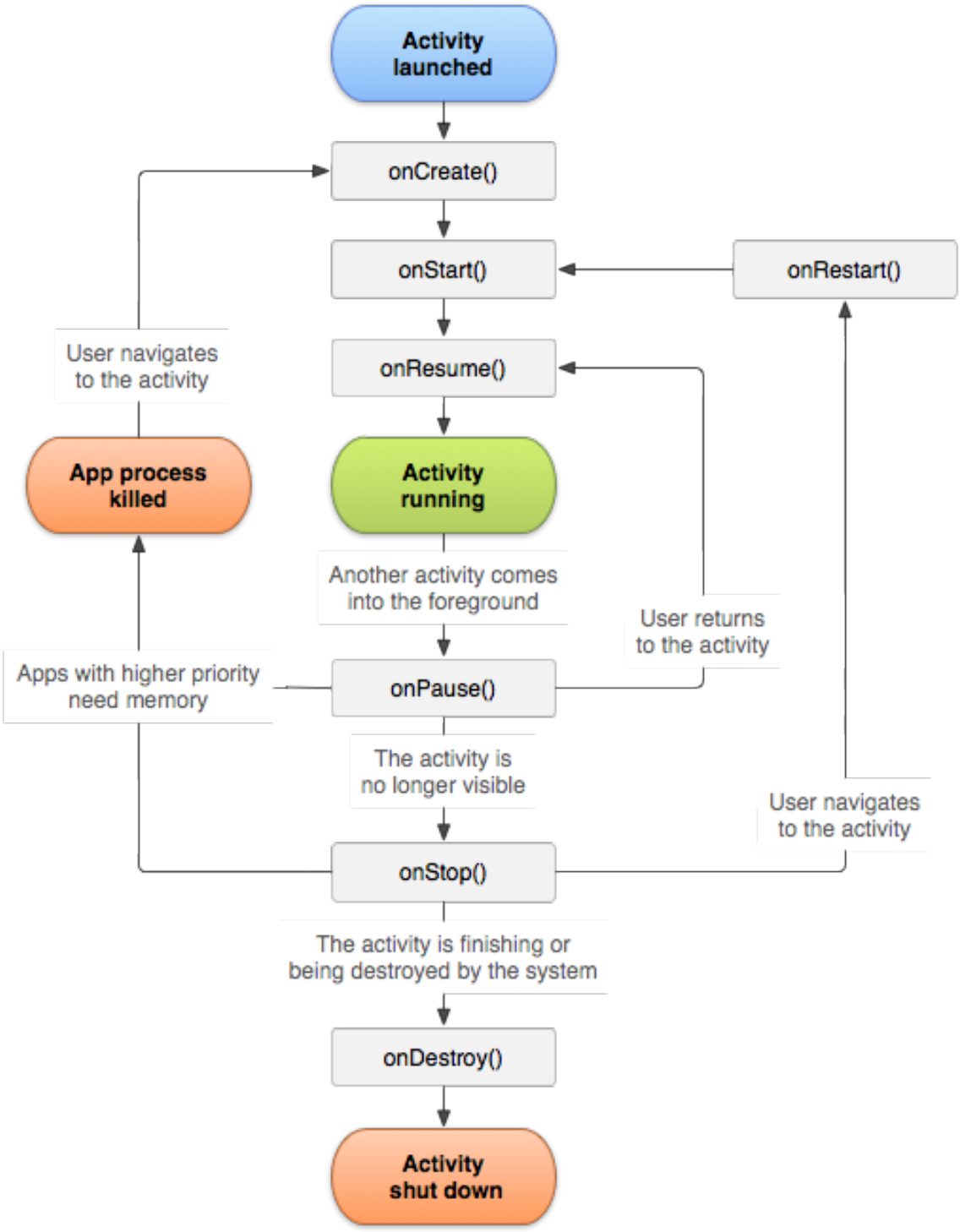
OpenCV je knihovna funkcí a algoritmů pro práci s počítačovým viděním, zpracováním obrazů a numerickými algoritmy s otevřeným zdrojovým kódem. Knihovna poskytuje nástroje pro zpracování a analýzu obsahu obrázků, včetně identifikace objektů na fotografiích (například osob a osob, textu, atd.), Sledování objektů, transformace obrazu, aplikace metod strojového učení a objevování společných prvků na různých úrovních obrázků.

OpenCV je implementován v C/C++ a je také vyvíjen pro Python, Java, Ruby, Matlab, Lua a další jazyky. Může být volně používán pro akademické a komerční účely - distribuován podle podmínek licence BSD. Má velkou komunitu a dobru dokumentaci. V současné době OpenCV je jednou z nejpopulárnějších knihoven pro zpracovaní obrázku, a má více než 5 milionů stažení, byla taký nedávno navržena jako základ pro standard Khronos v počítačovém vidění.

OpenCV obsahuje mnoho primitivních datových typů. Tato data nejsou z hlediska programovacích jazyků na vysoké úrovni primitivní, ale z pohledu OpenCV jsou nejzákladnější. Takové typy jako:

- Point je nejjednodušším typem dat. Tato jednoduchá struktura se skládá pouze ze dvou polí x a y typu int. Slouží k nastavení a zpracování grafických bodů.
- Size obsahuje dvě pole width a height typu int. Tato struktura slouží pro ukládaní informace o velikosti pole.
- Rect obsahuje čtyři pole x, y, width a height typu int a představuje obdélník.
- Scalar obsahuje čtyři proměnné double. Ve skutečnosti, Scalar zahrnuje jedno pole, který je ukazatelem na pole, které obsahuje čtyři čísla double. Struktura Scalar slouží pro ukládání dat o barvě objektu.

Kromě toho OpenCV obsahuje implementaci různých algoritmů pro extrakci příznaků, nejpopulárnější s nich jsou: SURF, ORB, SIFT, BRIEF.



Obrázek 3.1: Životní cyklus Android aplikace[3]

Kapitola 4

Návrh řešení

V této kapitole se budeme zabývat návrhem aplikace, která, by byla schopna rozpoznávat umění podle obrázku z kamery. Cílem aplikace je zjednodušit život milovníkům umění v galerii, a pomoci neuronové sítě a rozpoznávaní obrazů se dozvědět o zajímavém umění víc. Takže aplikace bude splňovat příjemné, rychlé, intuitivní ovládaní, aby si uživatel byl schopen od prvního startu aplikace bez problému začít ji používat. Začínat aplikace bude se zjištění polohy uživatele. Pak uživatel bude mít možnost nasměrovat mobil na zajímavé umělecké dílo. Pokud dílo bylo nalezeno, uživatel ihned dostane odpověď od aplikace s popisem díla a může v klidu přečíst nějaké nové zajímavé fakty o umění.

4.1 Zjištění polohy

Jak už bylo zmíněno dřív, aplikace bude začínat oknem zjištění polohy. A je to pro zvětšení přesností vyhledávaní, protože aplikace na základě polohy omezí počet obrázků. To znamená že galerie, která poskytla informace o svých uměleckých dílech dostane unikátní ID v systému, podle kterého modul rozpoznávaní vyhledá určitý klasifikátor vytrénovaný jenom na obrazcích z teto galerie.

Existuje několik způsobů zjištění pozice uživatele. První způsob je se pomocí GPS dozvědět souřadnice uživatele, tato metoda je víc přesnější a je lepší z pohledu uživatelského pohodlí, protože předpokládá miň zásahu od uživatele.

Druhý způsob je zjištění polohy od samotného uživatele, tzn. že uživatel samostatně nějakým způsobem dá aplikací vědět svou polohu. Tento způsob je taky dost přesný, ale není úplně důvěryhodný, protože uživatel může dát aplikaci úplně jinou adresu. Ale je vhodný pro případy kdy uživatel má problémy s GPS modulem, nebo ho z nějakých důvodů vůbec nechce zapínat.

Pro lepší uživatelskou zkušenost aplikace použije oba způsoby. Druhý způsob musí byt jednoduchý a pochopitelný, aby uživatelé, které mají malou zkušenost v použití mobilních zařízeni (například důchodci), mohli bez problémů použit aplikaci, z takového pohledu je nutně vybrat něco co by šlo naskenovat, například QR kód, který bude obsahovat zakódovaný unikátní hash kód galerie.

Celý uživatelský tok pro první okno bude vypadat jako na dalším obrázku 4.1. Popis toku:

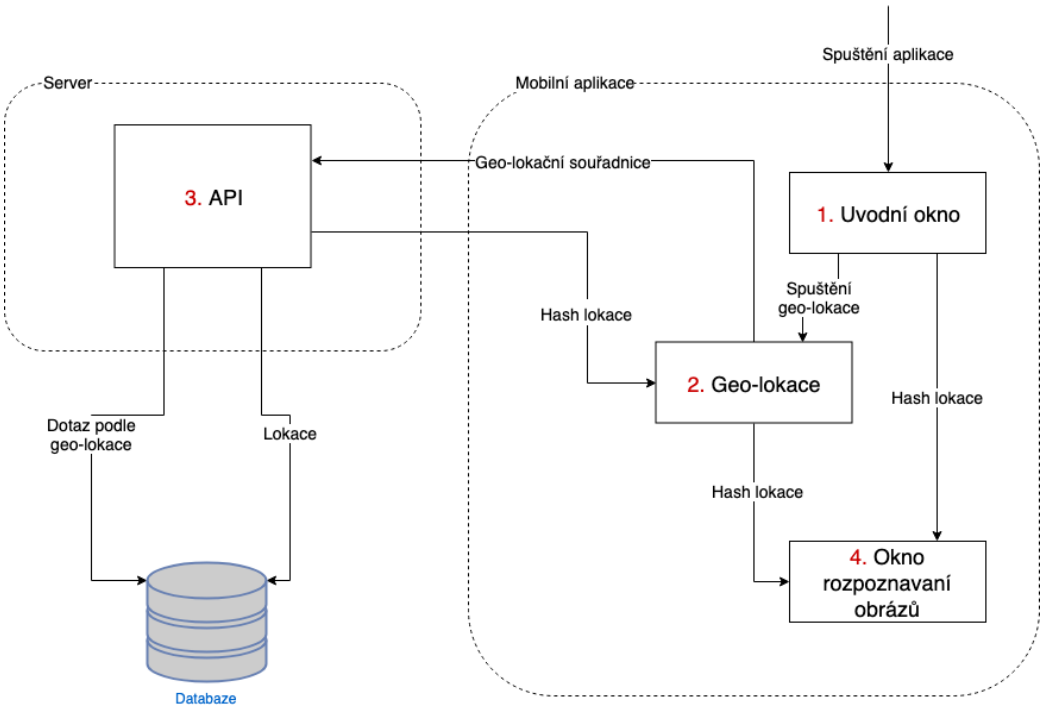
1. Při spuštění aplikace uživatel uvidí úvodní okno, které zkusí načíst QR kód s hashem polohy a zároveň zkusí spustit geo-lokaci (pokud GPS modul na smartphonu je za-

pnutý a má povolení) pro souběžní zjištění lokace. Pokud uživatel načte QR kód tak úvodní okno spustí hlavní okno a předá mu hash lokace (krok 4).

2. Modul geo-lokace zjistí polohu a pošle koordináty na server pomoci Rest API. Server pak vrátí buď hash lokace, nebo nic.

3. Server najde v databázi lokace s přesností 500 metrů a vrátí hash kód lokace klientovi.

4. Pokud hash lokace je zjištěn tak aplikace přesměruje uživatele na hlavní okno aplikace, které se zabývá rozpoznáváním obrazů.



Obrázek 4.1: Úvodní okno

4.1.1 Mockup

Úvodní okno vítá uživatele krátkým a lakonickým textem, který musí vysvětlit uživateli postup pro další použití aplikace 4.2. Taký okno má uprostřed čtverec, který musí zobrazovat skenovaní QR kódu ze zadní kamery.

4.1.2 API požadavek

Jako architektura pro spojení se serverem bude použita REST API. Tělo požadavku bude ve formátu JSON. Tabulka 4.1 popisuje požadavek na server na zjištění hash polohy. Pokud poloha nebyla zjištěná, server vrátí `null` jako odpověď.

Obrázek 4.2: Mockup úvodního okna

Metoda	POST			
Cesta	/api/localization			
Vstup	Název	Popis	Povinnost	Typ
	lat	Zeměpisná šířka	Povinné	string
	lon	Zeměpisná délka	Povinné	string
Vystup	Název	Popis	Typ	
	hash	Hash galerie	string	

Tabulka 4.1: Požadavek na zjištění hash polohy

Metoda	Kličové body	Čas(ms)	Čas/Bod
SIFT+SIFT	3665	5.989	0.0016341
SURF+SURF	3634	1.083	0.0002977
MSER+SIFT	323	0.889	0.0027533
BRISK+FREAK	466	2.531	0.0054322
BRISK+BRISK	466	0.235	0.0005046
ORB+ORB	500	0.236	0.0004715
FAST+BRIEF	11880	0.083	0.0000070

Tabulka 4.2: Porovnaní rychlosti algoritmů extrakce lokálních příznaků

4.2 Rozpoznávaní obrazů

Když hash kód lokace byl zjištěn úvodní okno přesměruje uživatele na hlavní okno, které se zabývá skenováním a pak dalším odesíláním na server pro rozpoznávání obrázku.

Existuje několik druhů skenovaní obrazů: skenovaní "na požádaní"(to znamená, že skenovaní a odesílaní proběhne jenom když uživatel například stiskne tlačítko nebo jiným způsobem dá jasně vědět aplikaci, že chce aby skenovaní proběhlo), kontinuitě skenovaní (aplikace bude skenovat a odesílat obrázek automaticky bez jasného aktivovaní ze strany uživatele).

Z ohledu na to, že aplikace musí být jasné pochopitelná a mít minimální a jasný design tak v tomto případě aplikace bude používat druhou metodu, což znamená redukovaný počet tlačítek.

Tato metoda znamená několik problémů:

1. Příliš velký počet požadavků na server
2. Celková velikost přenášených dat je příliš velká

4.2.1 Počet požadavků na server

Pro redukce požadavků je nutně vytvořit nějaký mechanismus který by měl automaticky kontrolovat nutnost poslat nový snímek na server.

Jako jednotku nutností lze považovat odlišnost v procentech mezí dvěma snímky. To znamená, že pokud snímek který už byl odeslán na server a nový snímek jsou dost odlišné tak je nutně poslat nový.o Kvůli tomu, že nový snímek může obsahovat stejný obrázek, ale z trochu jiného úhlu (tomu může způsobit uživatel, který ne může pořad byt ve stejné poloze), proto porovnaní pixel-by-pixel nedá dost přesný výsledek. Proto je nutné použit algoritmy na extrakci lokálních příznaku. Při volbě algoritmu je nejdůležitější náročnost na zařízeni, podle testovaní na OpenCV 2.4.8 na počítači z 3.4 GHz a 4 GB RAM, OS Windows 8 (tabulka 4.2), nejrychlejší algoritmus je detektor klíčových bodů FAST spolu z deskriptorem bodových příznaků BRIEF. Ale podle přesnosti ORB je mnohem lepší. Během testovaní různých zorných uhlů, rozostření obrazu, různého osvětlení, ukázal se jako jeden z nejpřesnějších, proto ohledně průměru rychlost/přesnost je nejlepší a během vývoje použiji právě ten.

Pak po nalezení vektoru příznaků je nutné najít vzdálenost mezí ním a předchozím vektorem. A pokud se snímky liší víc něž o 25% v tomto případě klient pošle nový snímek na server.

4.2.2 Celková velikost dát

Zmenšení počtu požadavku taký zmenší celkovou velikost dát, ale kvůli tomu že velikost obrázku který aplikace pošle je dost velká je nutné ho předzpracovat, což taký zvětší rychlost odezvy serveru, kvůli tomu, že server už nemusí předpracovávat obrázek.

Maximální rozlišení vstupního obrázku pro MobileNet je 224x224x3, proto předtím něž poslat obrázek je vhodné předem zmenšit rozlišení.

4.2.3 Uživatelský tok

Z ohledem na všechny požadavky od hlavního okna, uživatelský tok bude vypadat jako na dalším obrázku 4.4a.

1. Celý cyklus začíná tím, že snímek ze zadní kamery se pak pošle na ověření nutnosti odesílaní na server a na obrazovku.
2. Aplikace na základě příznaků porovná snímek z předchozím odesláním snímkem a pokud odlišnost je větší něž 25% tak předá obrázek na předzpracovaní.
3. Ve třetím kroku aplikace zmenší velikost a počet kanálů do 224x224x3.
4. Předzpracovaný obrázek pak modul pro odesílaní překóduje do base64 řetězce pro vyhovování RestAPI standardům a pošle na server.
5. Server obrázek přepošle na modul rozpoznávaní, který udělá extrakci příznaků a na její základě zjistí ID díla v databázi. Pak pošle dotaz na databázi a vrátí v API informace o dílu. Pokud se nepodařilo zjistit ID díla, modul vrátí `null`. Výsledek prací modulu server jako odpověď pošle zpátky klientovi.
6. Obrazovka zobrazí snímek s kamery nezávisle na požadavku a pokud server vrátil nějakou informace zobrazí ji jako na obrázku 4.4a.

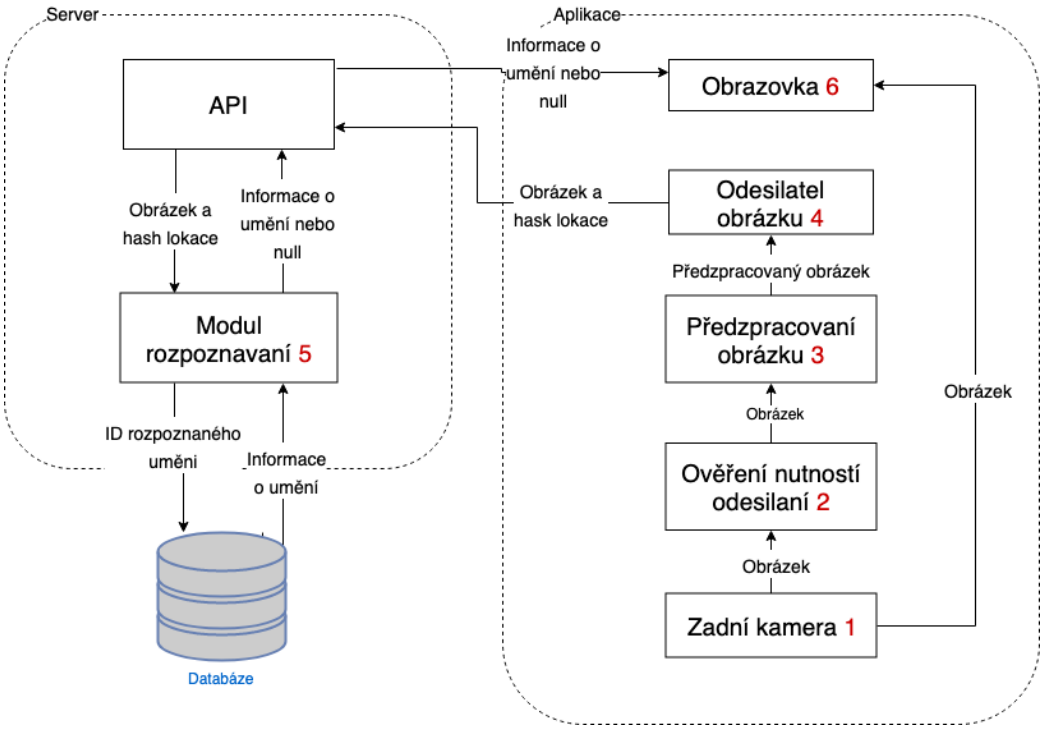
4.2.4 Mockup

Hlavní okno aplikace vítá uživatele náhledem se zadní kamery a textem který musí vysvětlit uživatelovi co dělat dál, mockup okna lze vidět na obrázku 4.4a.

Pokud snímek s kamery byl rozpoznaný serverem, na obrazovce se zobrazí panel s názvem díla a jménem autora, jako na obrázku 4.4b. Pak se uživatel může zatáhnout prstem panel nahoru, což zobrazí celý detail díla, včetně roku vytvoření a podrobnějšího popisu. Příklad lze vidět na obrázku 4.4c.

4.2.5 API požadavek

Jak už bylo zmíněno víš aplikace pro odesílaní požadavků používá formát JSON. Klient kromě snímku z kamery odesílá taký hash lokace. V tabulce 4.3, lze vidět detailnější popis dát které požaduje požadavek. Jako odpověď ze serveru aplikace dostane objekt `artwork` s popisem díla (atributy které obsahuje objekt lze vidět v tabulce 4.4) nebo `null`.



Obrázek 4.3: Hlavní okno

Metoda	POST			
Cesta	/api/recognize			
Vstup	Název	Popis	Povinnost	Typ
	image	Snímek	Povinné	string
	hash	Hash galerie	Povinné	string
Vystup	Název	Popis		Typ
	artwork	Informace o uměleckém díle		object | null

Tabulka 4.3: Požadavek na rozpoznávaní snímku

Název	Popis	Typ
id	Unikátní identifikační číslo díla	string
title	Název díla	string
description	Podrobnější popis díla	string
author	Autor díla	string
genre	Žánr díla	string
year	Rok vytvoření	integer

Tabulka 4.4: Popis objektu artwork

(a) Mockup hlavní obrazovky aplikace

(b) Mockup hlavní obrazovky aplikace s detailem díla

(c) Mockup detailního popisu díla

4.2.6 Modul rozpoznávaní

Jak už bylo zmíněno na vyhledávaní podobného obrázku bude použitý předem trénovaný model MobileNet, ale místo výstupní vrstvy, pro extrakce příznaků je nutné přidat vrstvu, která spojí příznaky ze vnitřních vrstev do jednoho vektoru.

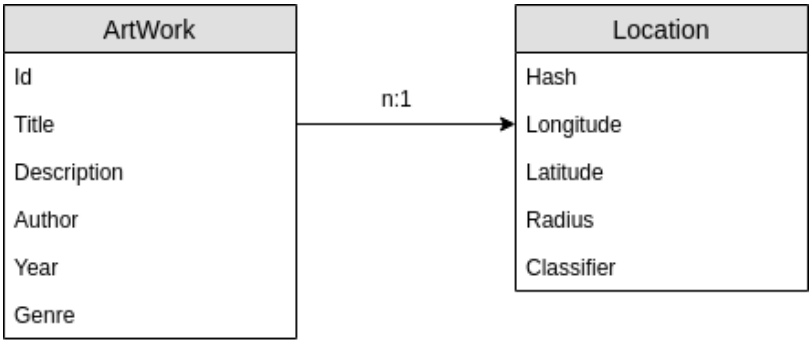
Pro zjištění nejvíc podobného obrázku na základě vektoru příznaků je nutné použit jeden z klasifikačních modelů. Nejjednodušší varianta k-nearest neighbours classifier, který porovná vektor z každým vektorem z vlastního seznamu a na základě vzdálenosti najde nejpodobnější obrázek. Problém algoritmu je v tom, že časová náročnost bude růst v závislosti s počtem obrázků v DB. Což znamená, že v tomto případě bude lépe použít jeden z modelů Supervised Machine Learning, to znamená algoritmus který naučí funkce mapovat vektor příznaků na nějaký určitý vystup. Jeden z takových algoritmů je Multiclass classification Logistic Regression, který je rozšířením Logistic Regression o víc tříd.

Klasifikátor Logistic Regression dostane na vstup vektor příznaků a vrátí ID nejvíc podobného obrázku. Pokud podobnost je větší něž 75%, modul použije ID na vyhledávaní informace v databáze.

4.3 Ukládaní dát do DB

Obrázek 4.5 obsahuje ER diagram databáze. V ER poloha je zvláštní entitou a obsahuje souřadnice galerie, unikátní hash, radius (v metrech), který pro rychlost a jednoduchost vypočtu server použije pro definování hranice místa. Classifier - je cesta do natrénované modele pro určitou galerii.

Entita ArtWork obsahuje základní informace o umění, unikátní ID, podle kterého modul rozpoznávaní bude vyhledávat umělecké dílo v DB, název díla, autor, rok tvorby a žánr.



Obrázek 4.5: ER diagram

Kapitola 5

Implementace

Tato kapitola se zaměřuje na výslednou implementaci systému pro rozpoznávaní umění. Většina kapitoly pojednává o detailech implementace systému a problémů, které se objevili během vývoje. Na konci kapitoly bude uveden stručný popis typického případu použití. Systém je implementovaný v jazyce Java a Python. Celé řešení může být rozděleno na dvě častí: server a Android aplikace.

5.1 Použité technologie

Jak už bylo zmíněno v sekce 3.2.2 pro vývoj mobilní častí aplikace byl použitý operační systém Android a jako programovací jazyk byl použitý Java v Android Studio s použitím knihovny OpenCV verzi 3.4.3 na snímaní a zpracovaní snímků. Minimální Adnroid SDK verze je 21, což odpovídá verzi Android 5.0 Lollipop, podle statistiky od Google to znamená, že v dnešní době aplikace bude podporovat 88.9% procent Android zařízeni.

Server je implementovaný s pomocí Django frameworku verze 1.11.17 v jazyce Python. Django jako standardní databázi používá SQLite, která je populární a snadno použitelná relační databáze SQLite, která je vhodná pro řešení malých úloh, testování. Ale pro produkční server nejlíp použit PostgreSQL. Django dovolí snadno změnit databází, díky tomu, že má vestavěnou ORM (Objektově relační mapování) vrstvu.

Pro práci s neuronovou síti byla použita knihovna Keras.

Pro zavedení produkční verze serverů bylo použité jednotné rozhraní pro izolaci aplikací do kontejnerů - Docker. Jako webhosting byl využitý Amazon Elastic Compute Cloud.

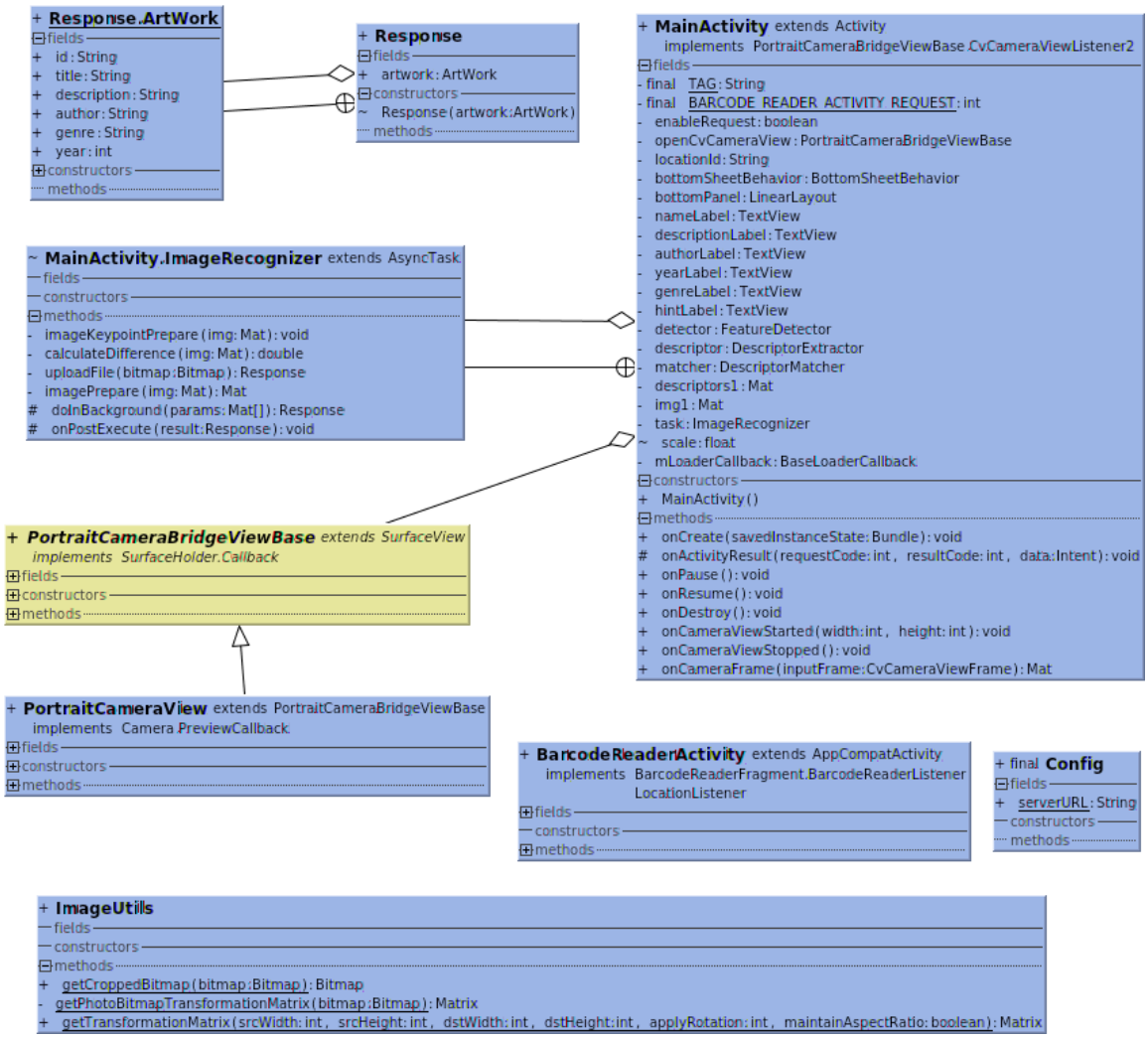
5.2 Struktura Android aplikace

Podle architektury z 4, aplikace lze rozdělit na dvě základní komponenty: server a klient (mobilní aplikace). A v teto sekce bude podrobnější probraná struktura a implementační detaily Android aplikace.

Digram tříd pro Android klienta vypadá jako ukázáno na obrázku 5.1.

5.2.1 MainActivity

MainActivity je hlavní třída aplikace a implementuje abstraktní třídu PortraitCameraBridgeViewBase.CvCameraViewListener2 která je popsaná níž. Na začátku jde inicializace všech OpenCV detektorů a descriptorů během definovaní BaseLoaderCallback objektu. Jak



Obrázek 5.1: Diagram tříd Android aplikace

už bylo popsáno v subsekce 4.2.1 pro extrakce příznaku a klíčových bodů byl použitý algoritmus ORB.

mLoaderCallback je objekt třídy BaseLoaderCallback, který existuje pro manipulace a inicializace komponentů které potřebuji OpenCV, a volá se po nahraní OpenCV knihovny. Inicializace deskriptorů ORB a povolení kamery proběhne jenom když status nahraní knihovny bude `LoaderCallbackInterface.SUCCESS`.

V metodě **onCreate** probíhá začáteční nastavovaní hodnot pro vnitřní atributy včetně nastavovaní proměnných pro práci s `Android Layout View` komponenty a posluchači na `BottomSheetBehavior`. `BottomSheetBehavior` je třída se standardní podporni knihovny Android SDK `android.support.design`, která je použitá pro přidaní aplikaci dolního sliding panelu.

onPanelStateChanged je metoda v posluchači `BottomSheetBehavior.setBottomSheetCallback`, která se vyvolá když se stav dolního sliding panelu změní. A v případě pokud panel bude otevřený nebo uživatel ho bude táhnout, posílaní žádosti na server se pozastaví (za to odpovídá proměna `enableRequest`).

Na konci metody `onCreate` se spouští aktivita `BarcodeReaderActivity` která je popsaná taký níž. Hlavní cíl teto aktivity vrátit naskenovaný hash kód s QR obrázku nebo s GPS. Hash kód je unikátním hashem lokace ve které se v dané době nachází uživatel.

onActivityResult je metoda která se volá když aktivita spuštěná pomoci metody `startActivityForResult`, vrátí nějaký výsledek. V tomto případě v teto metodě výsledek aktivity `BarcodeReaderActivity` (hash kód lokace) bude zapsaný do atributu `locationId`.

PortraitCameraBridgeViewBase.CvCameraViewListener2 je interface který popisuje metody pro poslouchaní kamery. **onCameraFrame** je jedna z popsaných v interfacu metod, a je zodpovědná za vyřizování obrázku z kamery. V těle metody probíhá zahájeni asynchronního úkolu (tasku) pro další předzpracovaní a posílaní obrázku na server a to jenom v případě, že už žádný task ne běží a posílaní požadavku je povoleno proměnou `enableRequest`.

onPause jak už bylo zmíněno v sekce 3.2.3, metoda se volá když přechází na pozadí a v tomto případě se zavolá `disableView` pro View OpenCV kamery, který vypne spojení s kamerou a pozastaví dodaní snímků.

onResume metoda, když aplikace přejde na popředí, zavolá inicializace spojení OpenCV s kamerou.

5.2.2 ImageRecognizer

ImageRecognizer je třída, která dědí třídu `AsyncTask` která je zodpovědná za provedení asynchroniích úkolů. `ImageRecognizer` v daném případě rozhoduje a posila obrázek na server pro rozpoznávaní, před tím provádí předzpracovaní pomocí OpenCV funkci.

doInBackground je metoda, která běží ve zvláštním vlákně a provádí předzpracovaní snímku (`imagePrepare`), pak porovná z předchozím odeslaným snímkem (`calculateDiffference`), pokud předchozí snímek neexistuje tak připraví ho (`imageKeypointPrepare`), a když odlišnost snímků je větší něž 25% tak odešlé požadavek s novým snímkem na server (`fileUpload`).

imagePrepare je metoda která slouží pro předzpracovaní snímku. Metoda mění velikost obrázku na 224x224 pixelů, pomocí OpenCV funkce `resize` z modulu `Imgproc`.

Pak pokud vzorový obrázek pro porovnaní není přepravený, zpracuje ho a zapíše vzorkový deskriptor do proměnné `descriptors1`. Pak spočítá rozdíl mezí vzorkovým obrázkem a aktuálním obrázkem (`calculateDiffference`).

Pokud rozdíl je větší něž 25% metoda obnoví vzorkový obrázek a pošle ho na server pomoci metody `fileUpload`.

calculateDiffference je metoda která zodpověděná za výpočet odlišností mezí snímky. Výpočet probíhá na základě distance mezi příznaky dvou snímků. Na začátku naklonuje původní obrázek, pak aby odhalil vliv barvy zkonvertuje ho na monochromní obrázek. Spočítá klíčové body pomocí metody `detect` objektu třídy `FeatureDetector` a na jejich základě deskriptory pomocí metody `compute` třídy `DescriptorExtractor`. Pak porovná typy matic (`Mat`) vzorkovacího a aktuálního obrázku, pokud náhodou ne shoduji tak se vrátí předem definované velké číslo, jinak spočítá vzdálenost, najde z poli vzdálenosti `matchesList` nejmenší vzdálenost a vrátí ji.

imageKeypointPrepare je metoda pro připravení a uložení příznaků vzorkovacího snímku. Příprava probíhá stejné jako na začátku metody `calculateDiffference` z tím rozdílem, že samotný obrázek a jeho příznaky se ukládají do rodičovských atributů.

fileUpload je metoda pro odesílaní požadavku se snímkem na server pro další rozpoznávaní. Metoda otevři spojení ze serverem pomocí metody `openConnection` třídy `URL`. A pak pomocí vytvoření objektu třídy `DataOutputStrem`, která se zabývá zapisováním dat

do toku. Po načtení odpovědí ze vstupního toku, metoda namapuje `JSON` data do objektu třídy `Response`, která je popsaná níž 5.2.5. Mapovaní probiha pomocí knihovny `Jackson`.

onPostExecute metoda na rozdíl od `doInBackground` běží ve stejném UI vláknu. Hlavní cíl metody je výsledek prací `doInBackground` zobrazit na obrazovku. V případě pokud server vrátil informace o rozpoznaném umění tak metoda dosadí jednotlivé atributy objektu do textových poli a pak zobrazí dolní sliding panel. Jinak pokud server vrátil `null`, metoda skryje sliding panel.

5.2.3 BarcodeReaderActivity

BarcodeReaderActivity je udělaná na základe knihovny `Android Barcode Reader`, která má licenci `Apache License 2.0`, má otevřený zdrojový kód a je založená na API `Google Mobile Vision`.

Třída `BarcodeReaderActivity` oproti příkladu na oficiální Github stránce knihovny, má drobné odlišností. V metodě `onCreate` aktivita spustí zjištění polohy podle GPS, pokud modul je zapnutý na mobilu a je povolený na použití.

Pokud GPS koordinaty byly zjištění tak aktivita pošle požadavek na server, který je popsaný v sekce 4.1.2. Pokud lokace byla zjištěna a server vrátil hash kód, tak aktivita dokonči práci a vrátí rodičovské aktivitě hash.

5.2.4 PortraitCameraView

PortraitCameraView a **PortraitCameraBridgeViewBase** jsou kopie tříd z knihovny OpenCV `JavaCameraView` a `CameraBridgeViewBase`, ale s drobnými odlišnostmi. Hlavní rozdíl je v tom, že původní třídy špatně pracuji na výšku, proto nutná byla modifikace aby snímek byl na celou obrazovku.

5.2.5 Response

Response je třída, cíl které je reprezentovat odpověď ze serveru. Jak už bylo popsáno dřív 5.2.2, pro mapovaní odpovědí ze serveru byla použitá knihovna `Jackson`, která pro správnost fungovaní `Response` musí obsahovat anotace, které popisuji propojení mezi atributy třídy a JSON objektu.

ArtWork je statická vnitřní třída která bude obsahovat mapovanou informaci o umění.

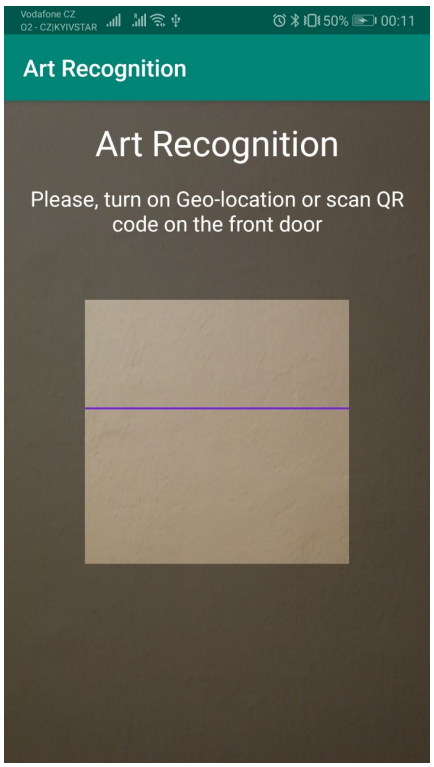
5.2.6 Snímky obrazovky

Okna už byly zvlášť podrobněji popsané v kapitole 4.

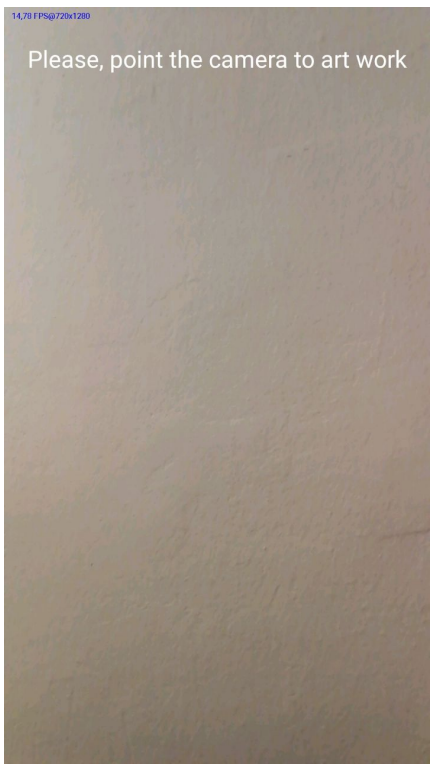
- Okno 5.2a odpovídá aktivitě `BarcodeReaderActivity` 5.2.3
- Okno 5.2b je hlavní okno které odpovidá `MainActivity` 5.2.1
- Na obrázku 5.2c zobrazený dolní panel, který uživatel může protáhnout nahoru prstem a uvidět detailní popis umění jako na dalším obrázku 5.2d.

5.3 Backend

Server a celý systém rozpoznávaní je napsaný v jazyce Python, pomoci knihoven Keras, OpenCV a frameworku Django. Kromě toho backend lze rozdělit na dvě části: příprava modelů (včetně dotrénováni a přetrénovaní) a samotný server.

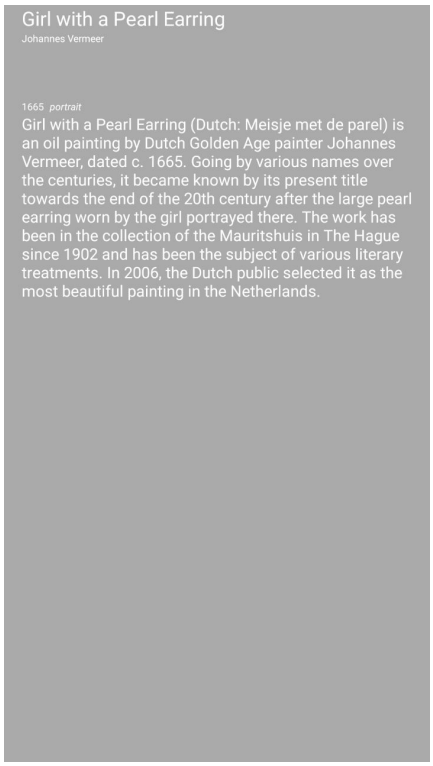


(a) Úvodní okno skenovaní QR kódu



(b) Hlavní okno aplikace

(c) Hlavní okno s detailem o umění



(d) Detailní popis díla

5.3.1 Příprava modelů

Samotnou přípravu také lze rozdělit na dva kroky:

- Příprava neuronové síti MobileNet pro další extrakce příznaků obrázku. Cílem teto častí je zvětšení pravděpodobnosti extrakce nutných příznaků, což zvětší unikátnost příznakových vektorů.
- Trénovaní klasifikátoru Logistic Regression na základě extrahovaných příznaků z neuronové sítí.

5.3.2 Příprava MobileNet

Použita v aplikací model MobileNet je založená na předem naučeném modelu s knihovny Keras. Model byl vycvičen na datasetu ImageNet, který obsahuje víc něž 14 milionů obrázků.

MobileNet byl změněn přidáním na konec vrstvy `Flatten`, která musí složit víc rozměrový vystup předchozí vrstvy do jednoho vektoru. Tato vrstva má název `custom`. Kvůli tomu, že poslední musí mít výstupní tvar `(2,)`, je nutné ji převést do takového tvaru, což lze udělat přidáním na konec ještě jedné vrstvy `Dense`.

Dál model byl dotrénován na fotkách umění z galerií a obrázcích díl které aplikace bude rozpoznávat.

Po dotrénováni poslední vrstva `Dense` bude smazaná kvůli tomu, že sloužila jenom pro účely správného vycvičení. A vystup vrstvy `custom` bude použitý jako vektor příznaků vstupního obrázku. Výsledný model má tvar jako v tabulce 5.1 (jsou zobrazení pár prvních vrstev a výstupní).

Vrstva (druh)	Výstupní tvar
input_1 (InputLayer)	(None, 224, 224, 3)
conv1_pad (ZeroPadding2D	(None, 225, 225, 3)
conv1 (Conv2D)	(None, 112, 112, 32)
conv1_bn (BatchNormalization)	(None, 112, 112, 32)
conv1_relu (ReLU)	(None, 112, 112, 32)
conv_dw_1 (DepthwiseConv2D)	(None, 112, 112, 32)
conv_dw_1_bn (BatchNormalization)	(None, 112, 112, 32)
conv_dw_1_relu (ReLU)	(None, 112, 112, 32)
conv_pw_1 (Conv2D)	(None, 112, 112, 64)
conv_pw_1_bn (BatchNormalization)	(None, 112, 112, 64)
conv_pw_1_relu (ReLU)	(None, 112, 112, 64)
conv_pad_2 (ZeroPadding2D)	(None, 113, 113, 64)
...	...
conv_dw_13 (DepthwiseConv2D)	(None, 7, 7, 1024)
conv_dw_13_bn (BatchNormalization)	(None, 7, 7, 1024)
conv_dw_13_relu (ReLU)	(None, 7, 7, 1024)
conv_pw_13 (Conv2D)	(None, 7, 7, 1024)
conv_pw_13_bn (BatchNormalization)	(None, 7, 7, 1024)
conv_pw_13_relu (ReLU)	(None, 7, 7, 1024)
custom (Flatten)	(None, 50176)

Tabulka 5.1: Výsledný tvar modelu pro extrakci příznaků

5.3.3 Příprava Logistic Regression

Během vývoje pro mapovaní unikátního ID umění na vektor příznaků byl použitý Logistic Regression model z open source knihovny `scikit-learn`, jako bylo popsáno v sekce 4.2.6.

Modul vycvičení Logistic Regression dostane výsledek extrakce příznaků z předchozího kroku, který je tříděný do zvláštních unikátních ID, které pak budou představovat ID klíč v databázi. Pak pomocí knihovny `pickle` natrénovaný model je uložený do souboru s příponou `.pickle`.

5.4 Problémy

V teto sekce jsou popsané zajímavé problémy, které vyskytly během vývoje aplikace a jejich řešení. Většina problémů vyskytla při napsaní aplikace pro Android.

5.4.1 Snímková frekvence

Bez žádné modifikace OpenCV kamera má snímkovou frekvence (FPS) 3-8 snímků za vteřinu, což je relativně velice málo proto bylo nutně toto číslo zvětšit. Existuje několik způsobů: opravit FPS parametry kamery, zmenšit velikost obrazovky. Je nutně vyzkoušet oba.

- Kamera v Android SDK podporuje několik parametrů, které mohou zvětšit rychlost přehledu snímku, za prve je nutně nastavit paramter `fast-fps-mode`, který je podporován některými smartphony a dovolí zvětšit horní omezení FPS. Pak je nutné zjistit podporované kamerou FPS rozsahy, a nastavit největší.

- Pro zmenšení rozlišení přehledu je nutně na začátku zjistit seznam podporovaných kamerou rozlišení, což obyčejné zahrnuje rozlišení: 720x540, 720x720, 1280x720 atd. Z uživatelského pohledu nejlepším rozlišením bude první rozlišení ze stranou větší něž 1000 pixelů.

Po těchto krocích se podařilo zvětšit FPS do 15-20 snímků/vteřina.

5.4.2 Zmizení dolního panelu

Během testovaní aplikace bylo pozorováno, že dolní panel se ne zobrazí. Po experimentech bylo zjištěno, že dolní panel se zobrazuje, ale pod plátnem snímku. Výzkum a vyhledání podobných problémů na webu, nevrátilo žádné výsledky. Bližší seznámeni se způsobem vykreslovaní snímku ukázalo, že metoda která na to použitá mění pořadí v `RenderNode`, což způsobí zobrazení panel na pozadí.

Pro opravení je nutné nastavit pro panel fixovaní z-index který bude zobrazovat panel vždy nad kamerovým plátnem.

Kapitola 6

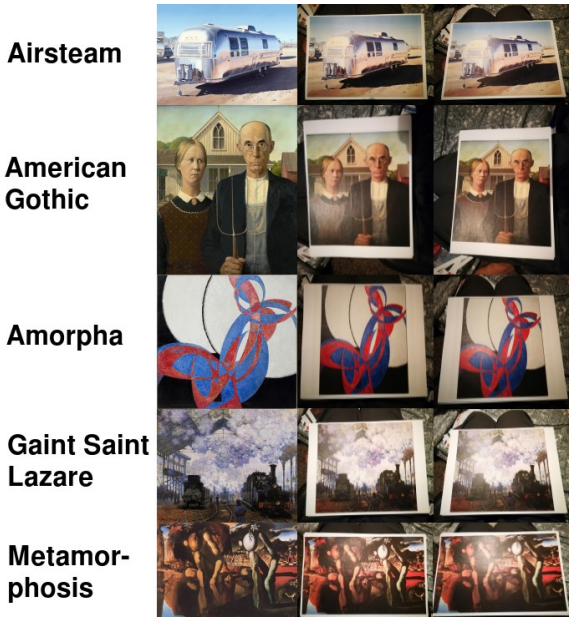
Výsledky

Tato kapitola popisuje testy, testovaní výsledné aplikace, výsledky těchto testů a jejich ohodnocení. Hlavním cílem testovaní bude zjištění klíčových parametrů: rychlosti, přesnosti a použitelnosti s uživatelského hlediska.

6.1 Návrh testovacích sad

Vzhledem k tomu, že testovaní musí probíhat na obrazcích, které budou co nejvíc podobné reálnému použití, proto bylo rozhodnuto použit 40 uměleckých díla, které byly vytištěné a pak ofocení pro další testy. Jeden z důvodů proč je nutně ofotit je také to, že dílo lze pak otestovat na různých pozadí a dozvědět závislost příznaků na samotném obrázku bez ohledu na polohu.

Díla byly vybrané různé od realismu do abstrakce. Díla obsahuji nejenom objekty reálného světa, ale i amorfní obrazy.



Obrázek 6.1: Ukázka testovacího datasetu

6.2 Návrh testovaní

Jak bylo zmíněno nahoře testovaní bude probíhat ve třech různých směrech: přesnost, rychlost a použitelnost. Proto testovaní taký je nutně rozdělit na tři části:

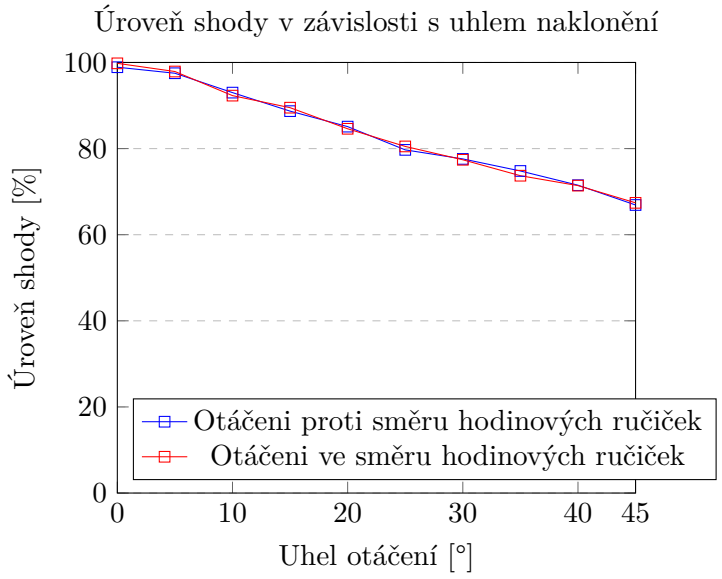
1. **Přesnost** je taký dost komplexní pojem, proto aby ověřit vliv různých faktorů je nutně rozdělit toto testovaní na další podčásti:
   - **Otáčeni** - testovaní uhlu otáčení díla na ploše, maximální uhel bude 30° protože kvůli tomu, že ve světě umění dílo pod větším uhlem může znamenat úplně zvláštní práci.
   - **Naklonění** - testovaní naklonění kamery relativně ploše.
   - **Osvětlení** - testovaní osvětlení díla. Úroveň osvětlení bude v jednotka intenzity osvětlení - lux. V galerii osvětlení může lišit, ale ve většině maximální úroveň osvětlení je 300 lux.[12]
2. **Rychlost** bude otestovaná na dvou zdroji domácím počítací a produkčním serveru na AWS.
3. **Použitelnost** bude otestovaná na zavřené kontrolované skupině uživatelů ve tvaru alfa-testovaní.

6.3 Výsledky testů

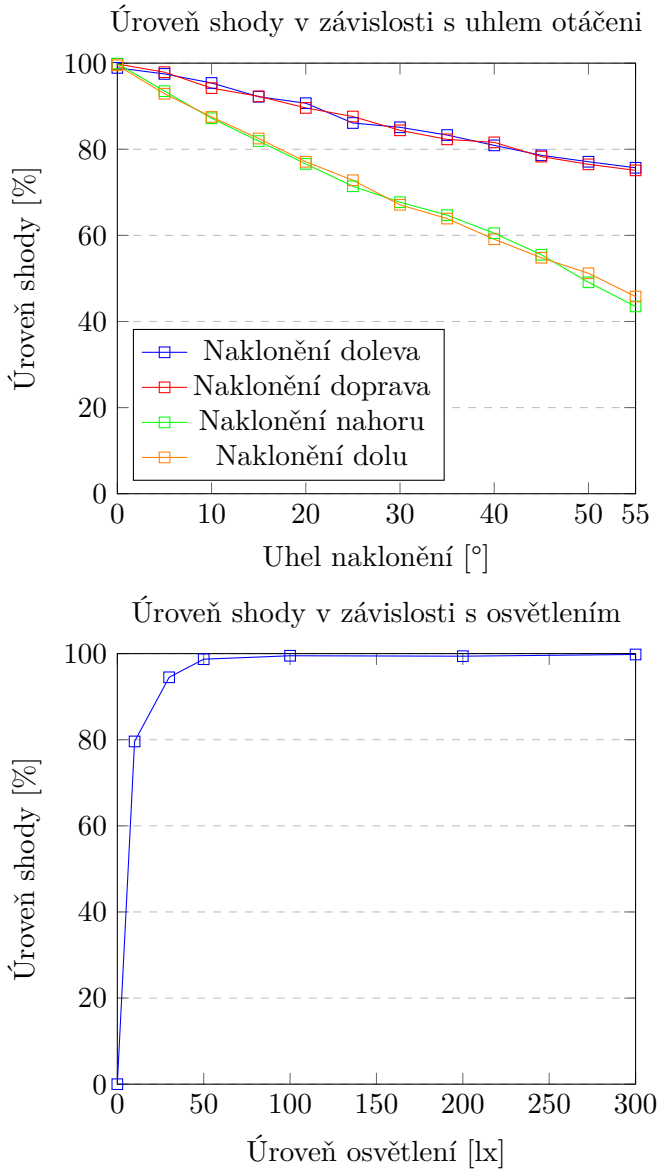
Tato sekce je rozdělená na častí podle návrhu v předchozí.

6.3.1 Přesnost

Prvním bude otestovaná přesnost. Níž lze vidět výsledky testů otáčeni ve směru hodinových ručiček a proti. Diagramy ukazuji závislost míru shody s uhlem otáčeni. Úroveň shody použitý pro sestavení diagramu je průměrem úrovní pro určitý uhel.



Další diagram zobrazuje závislost shody na uhlu naklonění.



6.3.2 Rychlost

Testovaní rychlosti proběhlo na počítací Lenovo ThinkPad, system Fedora 27, Intel Core i7-6600U CPU, 2.60GHz, paměť 19.0 GiB. Během testovaní nejpomalejší součástkou byla extrakce příznaků, která byla v rozmezí 40ms - 100ms, nejrychlejší byla model Logistic Regression, čas které byl v rozmezí 1ms-4ms.

Instance na AWS má CPU Intel Scalable 3.3GHz, paměť 1.0 GiB. Ve produkce rozpoznávaní je celkem pomalejší rozmezí pro extrakce stalo 100ms - 200ms a Logistic Regression je 3ms - 10ms.

6.3.3 Použitelnost

Testovaní použitelnosti probíhalo na malé skupině lidí, zúčastnilo se 4 osoby. Účastníky dostali několik běžných, pro uživatele aplikace, úkoly:

1. Spustit aplikace a načíst QR kód
2. Rozpoznat dílo
3. Otevřít podrobnější informace o díle

Po splnění úkolu, každý účastník dostal formulář s následujícími otázkami:

- Ohodnotíte pochopitelnost aplikaci (1-5), kde 1 - vůbec nepochopitelná.
- Ohodnotíte samotné rozpoznávaní (1-5)
- Ohodnotíte rychlost odezvy (1-5)
- Používal/-a byste tuto aplikaci v galerii? (Ano/Ne)
- Váši další komentáře a požadavky.

Podle zpětné vazby: pochopitelnost - 4, rozpoznávaní - 3.75, rychlost - 4.25. Většina komentářů byla o tom, že občas aplikace nechce rozpoznávat dílo a to že slide vlastnost dolního panelu musí být nějak označená navíc pro lepší pochopitelnost. Ale v libovolném případě 4 z 5 osob odpověděli, že by používali aplikace ve skutečných galerii.

6.4 Shrnuti

Celkem rozpoznávaní na základě dotrénováného MobileNet modelu a Logistic Regression se ukázalo dobře. Aplikace ne jenom ne zavázaná na určitém pozadí, ale i na objektech reálného světa. Test na osvětlení ukázal, že aplikace může v pohodě fungovat v běžných galerii a různé testy uhlů to, že skoro s libovolné polohy před obrázkem aplikace může naskenovat a rozpoznat dílo.

Samozřejmě aplikace není ideální, během uživatelského testu vyskytly nějaké chyby, které už byly popsané nahoře, což je nutně opravovat v dalších verzích aplikace.

Kapitola 7

Závěr

Cílem diplomové práce bylo vytvořit aplikace, která by byla schopna na základě snímku se zadní kamery rozpoznat umělecké dílo. Kromě samotné aplikace pro Android bylo také implementováno server, hlavním cílem kterého je nejenom uložení informace o umění, ale rozpoznávaní.

Během vývoje podařilo naprogramovat fungující systém pro rozpoznávaní obrazů na základě konvoluční neuronové síti, s jednoduchým a pochopitelným uživatelským rozhraním. Aplikace byla otestovaná na vytištěných uměleckých dílech, což přibližuje testovaní do reáliích produkčních podmínek.

Testy ukázali jako silné stránky řešení tak odhalili chyby a zlepšení které budou provedené v dalších verzích aplikace. Kromě opravy chyb a uživatelského rozhraní jsou spousta nápadu jak lze zlepšit aplikace a jaké funkce bylo by super přidat, jako například nějaké začátky sociální síti.

Pracovat nad tímto projektem bylo zajímavé a je v planu pokračovat vývoj, aspoň do první stabilní beta verze.

Literatura

[1] Abdelmgeid, H. A., Aly Amin; Alshazly: *Image Features Detection, Description and Matching*. Springer International Publishing Switzerland, 2016, [Online].
URL https://pdfs.semanticscholar.org/322f/f387087134ac776a4270cd55e7f3334edeb2.pdf

[2] Chollet, F.: *Xception: Deep Learning with Depthwise Separable Convolutions*. arXiv, Duben 2017, [Online].
URL https://arxiv.org/pdf/1610.02357.pdf

[3] Developers, G.: Understand the Activity Lifecycle. 2017, [Online].
URL https://developer.android.com/guide/components/activities/activity-lifecycle

[4] ESAAK, S.: *What Are the Visual Arts?* ThoughtCo, Leden 2019, [Online].
URL https://www.thoughtco.com/what-are-the-visual-arts-182706

[5] Foundation, D. S.: Django Documentation. 2019, [Online].
URL https://buildmedia.readthedocs.org/media/pdf/django/2.2.x/django.pdf

[6] Howard, A. G.; Zhu, M.; Chen, B.; aj.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017, 1704.04861.
URL http://arxiv.org/abs/1704.04861

[7] IDC: Smartphone Market Share. 2019, [Online].
URL https://www.idc.com/promo/smartphone-market-share/os

[8] Şahin Işık; Kemal Özkan: *A Comparative Evaluation of Well-known Feature Detectors and Descriptors*. International Journal of Applied Mathematics, Electronics and Computers, Srpen 2014, [Online].
URL http://dergipark.gov.tr/download/article-file/89429

[9] LECUN, Y.: *Deep Learning Tutorial*. Talks by Members of CBLL, Červen 2013, [Online].
URL http://www.cs.nyu.edu/~yann/talks/lecun-ranzatoicml2013.pdf

[10] Nair, V.; E. Hinton, G.: Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair. 06 2010.

[11] Smith, S. W.: The Scientist and Engineer's Guide to Digital Signal Processing. 1997.

[12] Sylvania, F.: LIGHTING FOR MUSEUMS AND GALLERIES. 2015, [Online].
URL https://www.sylvania-lighting.com/documents/documents/Museums%20and%20Galleries%20-%20Brochure%20-%20English.PDF

[13] VOLNÁ, E.: *Neuronové sítě 1.* Ostravská univerzita v Ostravě, 2008, [Online]. URL http://www1.osu.cz/~volna/Neuronove_site_skripta.pdf

Příloha A

Obsah CD

Přiložený CD obsahuje následující položky:

backend - Django server
application - Android aplikace
scripts - podporové skripty
master_thesis.pdf - text diplomové práce v elektronické podobě
video.mp4 - video k aplikaci
README - README soubor