Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Klasifikátor architektonických slohů ze snímků budov

Plzeň 2022 Daniel Schnurpfeil

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Daniel SCHNURPFEIL**
Osobní číslo:	**A18B0312P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Klasifikátor architektonických slohů ze snímků budov**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s technikami klasifikace snímků hlubokými neuronovými sítěmi, zejména v režimu využití optimalizovaných knihoven dostupných k integraci s vyvíjenou aplikací (např. PyTorch, DLib, TensorFlow).
2. Prostudujte (případně konzultujte s architektem) architektonické slohy, se kterými je možno se běžně setkat, s ohledem na vhodný návrh klasifikačních tříd, navrhněte a vypracujte postupy přípravy trénovacích dat klasifikátoru.
3. Naprogramujte aplikaci (mobilní nebo webovou) s jednoduchým uživatelským rozhraním, která umožní pořídit, resp. nahrát snímek budovy a provede jeho klasifikaci s cílem určit, o jaký architektonický sloh se jedná.
4. Vyhodnoťte vhodnost implementovaných postupů a úspěšnost klasifikace, otestujte praktickou použitelnost aplikace (např. scénářovým testováním).
5. Dosažené výsledky a získané poznatky a závěry pečlivě popište.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**
Rozsah grafických prací: **dle potřeby**
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce: **Ing. Kamil Ekštein, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **4. října 2021**
Termín odevzdání bakalářské práce: **5. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Doc. Ing. Přemysl Brada, MSc., Ph.D.
vedoucí katedry

V Plzni dne 14. října 2021

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů. V textu této práce jsou použity názvy programových produktů, firem apod., které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

V Plzni dne 4. května 2022

Daniel Schnurpfeil

Poděkování

Děkuji Ing. arch. Ladislavu Schejbalovi za dobré rady a vstřícnost při hledání vhodných architektonických slohů pro automatickou detekci. Dále děkuji Ing. Kamilu Ekšteinovi, Ph.D za jeho odborné vedení bakalářské práce a čas, který mi věnoval na konzultacích.

Abstract

The aim of this work is to develop an application that will classify a loaded image of the object (building) from a built-in mobile device or mobile camera into one of the architectural styles such as gothic, renaissance, baroque, art nouveau, functionalism, and others. The application will be designed for the mobile device or as a web application for accessibility over the Internet. The classifier used in the application will be based on convolutional neural network.

The greatest contribution of the work lies not in the application itself, but in the creation of a neural classifier for automatic building recognition that can be used or expanded in the future.

Abstrakt

Cílem práce je vyvinout aplikaci, která bude z načteného obrázku anebo za pomoci vestavěného fotoaparátu mobilního zařízení klasifikovat snímek objektu (budovy) do některého z architektonických slohů jako je například gotika, renesance, baroko, secese, funkcionalismus a další. Aplikace bude určena pro mobilní zařízení nebo jako webová aplikace dostupná přes internet. Použitý klasifikátor aplikace bude založen na technice konvolučních neuronových sítí.

Největší přínos práce nespočívá v aplikaci jako takové, ale ve vytvoření neuronového klasifikátoru pro automatické rozpoznávání budov, který může být v budoucnu také použit a dále rozšiřován.

Obsah

1 Úvod

Rozpoznávání a klasifikace objektů z fotografií zažívají ve dvacátých letech jednadvacátého století obrovský rozmach. Hlavně kvůli neustálému výzkumu zabývajícímu se neuronovými sítěmi, jehož důsledkem je využívání neuronových sítí v mnoha oblastech. Například ke klasifikaci architektonických slohů budov, která je tématem této práce.

Architektonické slohy jsou z historického hlediska velice důležité. Jejich prvky jsou klíčové pro budování staveb v budoucnu. Bohužel kromě lidí, kteří se o tento obor zajímají, by se našlo pouze několik architektů, kteří dokáží s jistotou poznat při pohledu na stavbu, o jaký sloh se jedná. Toto předchozí tvrzení je sice pouze domněnkou, nicméně by se ale hodilo vyvinout aplikaci, která dokáže klasifikovat tyto slohy.

Cílem práce je vytvořit aplikaci, která má výše zmíněné vlastnosti. Nejprve je třeba natrénovat vhodnou neuronovou síť získanými trénovacími daty a po dosažení uspokojivé úspěšnosti neuronové sítě naprogramovat aplikaci nebo aplikace, které využívají danou neuronovou síť za účelem zobrazení výsledku pro vstupní fotografii s konkrétní budovou.

V první části se budeme zabývat principy neuronových sítí pro klasifikaci obrázků a dostupnými optimalizovanými knihovnami, ze kterých jsou zmíněny PyTorch a TensorFlow. Dále se zaměříme na architektonické slohy, které jsou po konzultaci s architektem vhodné pro klasifikaci.

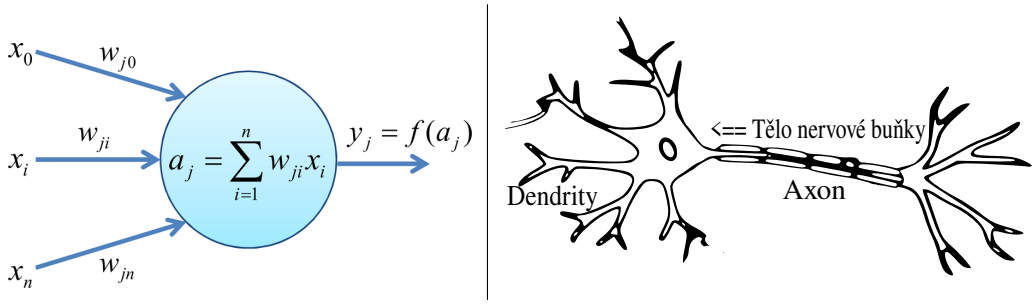
V druhé části jsou popsané postupy přípravy dat, které byly použity pro učení neuronových sítí. Následně jsou popsány použité neuronové sítě a zhodnocen jejich potenciál pro použití v aplikacích včetně výsledků jejich úspěšností. V neposlední řadě jsou také popsány implementace jednotlivých aplikací, skriptů pro přípravu dat a skriptů určených k naučení neuronových sítí.

2 Neuronová síť

V této kapitole se budeme zabývat problematikou umělých neuronových sítí, přičemž se omezíme pouze na neuronové sítě vhodné pro klasifikaci obrázků. Nejprve je ale třeba seznámit se základními principy neuronových sítí.

2.1 Neuron

V biologickém pojetí je to buňka nervové soustavy, která podle intenzity vstupních signálů může vést daný signál anebo signály dále. Umělý neuron funguje na podobném principu, kde jsou vstupní signály (dendrity v biologickém pojetí) reprezentovány číselnými hodnotami, které jsou spočteny pomocí vhodného matematického operátoru s takzvanými vahami v lineárním zobrazení (matematická funkce). Toto lineární zobrazení značí tělo nervové buňky a výsledek zobrazení je v biologickém pojetí axon vedoucí signál dále. Podrobněji je neuron ukázán na perceptronu, což je nejjednodušší neuronová síť sestávající se z jediného neuronu [22].

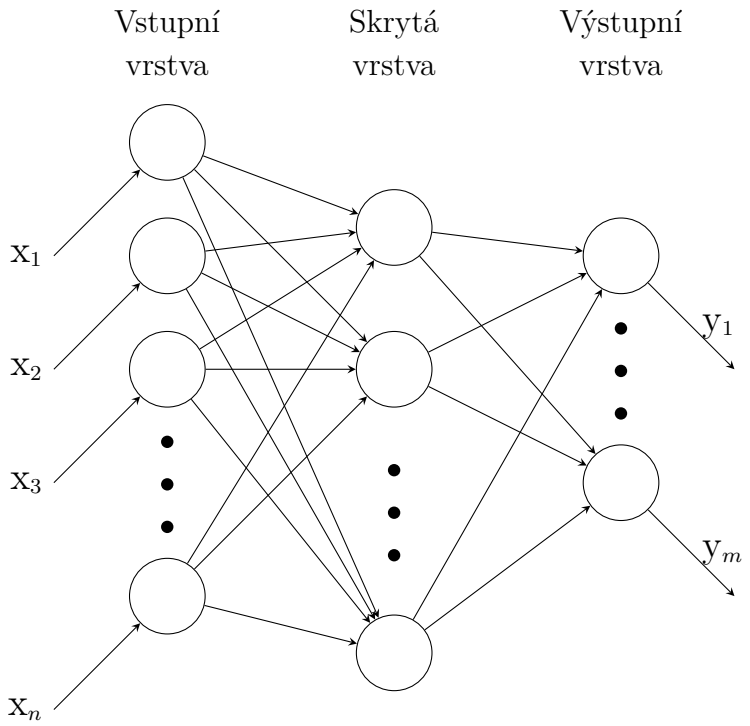


Obrázek 2.1: Vlevo: perceptron (obrázek převzat z [36]) a vpravo: neuron v biologickém pojetí

Z obrázku 2.1 (vlevo) je vidět, že se jedná o funkci (lineární zobrazení) množiny vstupů $\{x_0, x_1 ... x_n\}$ na výstup y_j. Každému vstupu je přiřazena váha w_{jn}. Dále se vypočte pomocí daného operátoru (v tomto případě suma) vstup do přenosové [22]/aktivační funkce, která může být různá. Například funkce sigmoid ze které je vypočtena hodnota výstupu y_j. Důležité je dodat, že výstup y_j je číslo v intervalu například od 0 do 1 a může být použito jako vstup do dalšího neuronu. Pokud je takto použito, tak se už nejedná o neuron ale o neuronovou síť.

2.2 Vrstevnatá neuronová síť pro klasifikaci

Neurony se většinou uspořádají do vrstev. Výstup neuronu z každé vrstvy kromě výstupní je vstupem do každého neuronu vrstvy následující, viz obrázek 2.2.



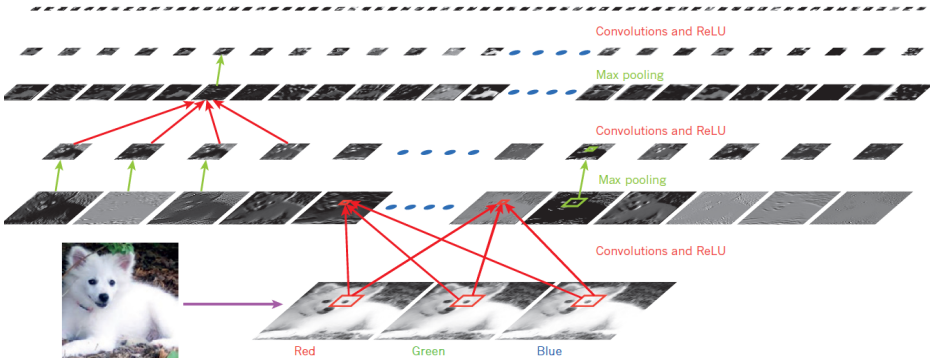
Obrázek 2.2: Nákres vrstevnaté neuronové sítě

Výstupní vrstvou je uspořádaná n-tice prvků v daném intervalu například od 0 do 1, kde každé prvek může indikovat (v případě aktivační funkce softmax indikuje) pravděpodobnost zda patří vstupní objekt do třídy. Tato n-tice je indexována. Ke každému indexu je přiřazena jedna třída. V našem případě je třídou architektonický sloh. Prvek s nejvyšší pravděpodobností je pokládán za výsledek, respektive výsledkem je třída přiřazená k indexu tohoto prvku.

2.3 Konvoluční neuronová síť

Konvoluční neuronová síť je určena ke zpracování vícerozměrných dat [20]. Takovými daty jsou například barevné obrázky, které mohou být reprezentovány například ze tří dvourozměrných polí obsahujících intenzity pixelů ve třech barevných kanálech (červená, zelená a modrá).

Nejprve však jednoduše popišme konvoluci. Nejjednodušeji řečeno, konvoluce je matematická operace, pomocí které se v našem případě upravuje obrázek například na obrázek, kde jsou zvýrazněny hrany, což je klíčové pro objekty, které tam hledáme.



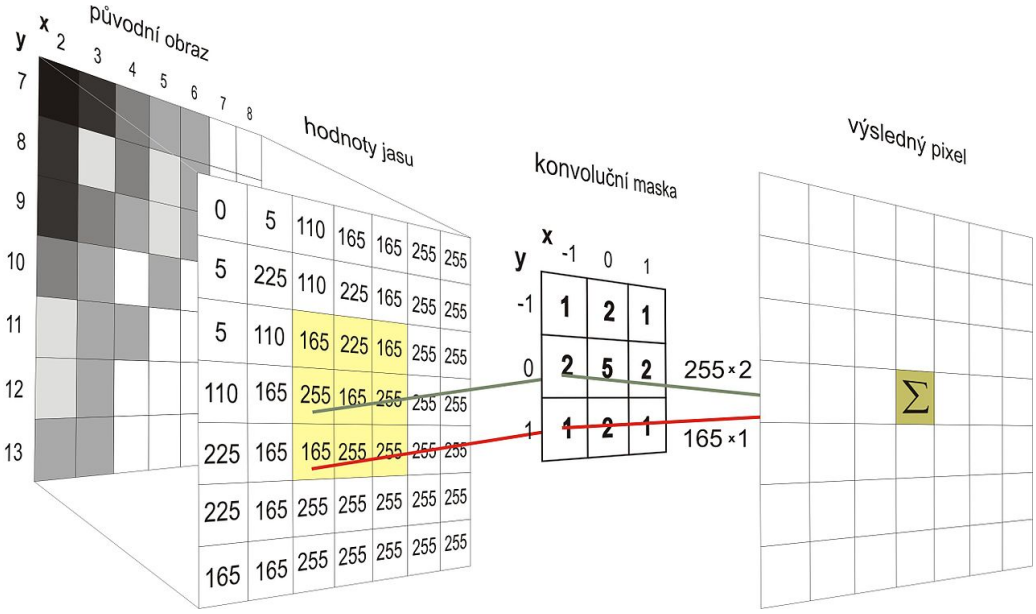
Obrázek 2.3: Vrstvy konvoluční sítě (obrázek převzat z [20])

2.3.1 Diskrétní maticová konvoluce

Nyní se podívejme na konvoluci podrobněji. Ku příkladu mějme černobílý obrázek reprezentovaný maticí, která je zobrazena vlevo na obrázku 2.4. Hodnoty v matici znázorňují intenzity jasu obrazových bodů (pixelů). Dále máme takzvanou matici jádra (konvoluční maska na obrázku 2.4). Obě matice se následně zpracují v následující funkci:

$$V_{i,j} = (M, N)_{i,j} = \sum_{a=-k}^{k} \sum_{b=-k}^{k} M(i-a, j-b) \cdot N(a, b), \tag{2.1}$$

kde $V_{i,j}$ je výsledná hodnota pixelu na pozici indexů i a j, M je oblast v matici V a N je matice jádra o k řádcích a k sloupcích. Diskrétní maticová konvoluce je vidět na následujícím obrázku 2.4.

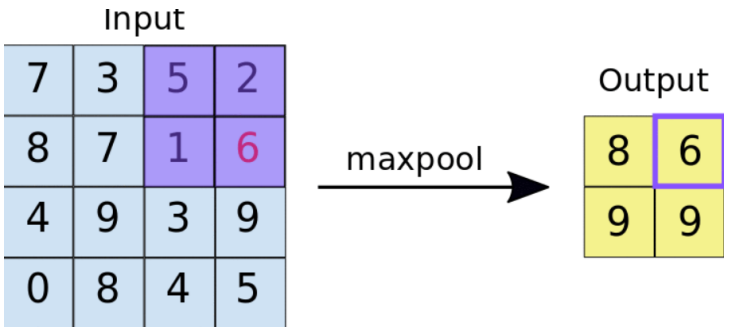


Obrázek 2.4: Příklad konvoluce (obrázek převzat z [8])

Z obrázku 2.4 je již zřejmé, že pro matici jádra o třech sloupcích a třech řádcích se násobí každá hodnota indexu s hodnotou indexu dané oblasti matice se stejnými rozměry jako matice jádra.

2.3.2 Podvzorkování obrázku (Pooling)

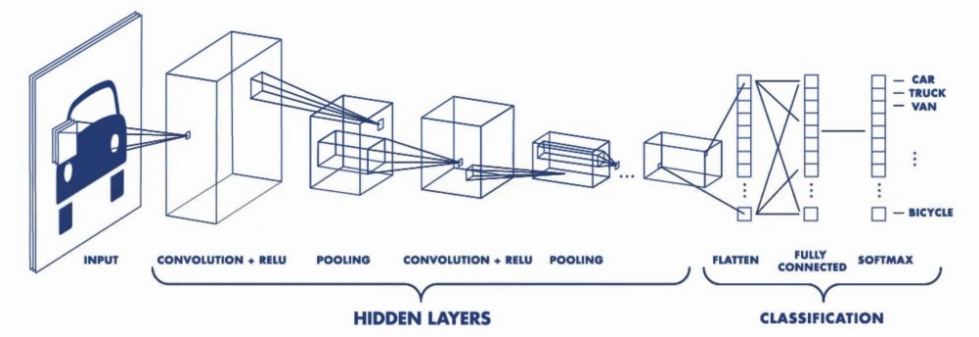
Tato metoda se podobá konvoluci popsané v minulých odstavcích sekce 2.3.1. Metoda prochází matici obrázku po oblastech (například dvakrát dva pixely popsané na obrázku 2.5) a vypočte právě jeden pixel pomocí definované funkce například průměrováním hodnot v daných oblastech (average pooling), nebo vypočtením maximální hodnoty v každé podoblasti (max pooling).



Obrázek 2.5: Příklad podvzorkování pixelů v obrázku (převzato z [32])

2.3.3 Architektura konvoluční neuronové sítě

Struktura konvoluční neuronové sítě je složena ze dvou částí. První je část, která zpracovává vstupní obrázek. Tato část se skládá z konvolučních a pooling vrstev. Výsledkem této části je vektor příznaků, který se použije jako vstup do vrstevnaté neuronové sítě (popsaná na obrázku 2.2). Celá architektura je popsána na obrázku 2.6.



Obrázek 2.6: Architektura celé konvoluční neuronové sítě (obrázek převzat z [17])

Konvoluční sítě se drží čtyř zásad. První zásadou jsou plně propojené vrstvy (každý neuron jedné vrstvy je propojen s každým neuronem vrstvy následující), dále je typické použití mnoha vrstev neuronů, viz obrázek 2.2, pak je to vzorkování popsané v minulé sekci 2.5, a nakonec sdílené váhy, což znamená, že každý filtr v dané vrstvě (viz obrázek 2.3) obsahuje stejnou váhu (popsáno v sekci 2.1). Sdílené váhy mají výhody. Například u dat v poli, jako jsou obrázky, jsou lokální skupiny hodnot často vysoce korelované a vytvářejí charakteristické lokální hodnoty z motivů, které lze snadno detekovat a jsou neměnné v závislosti na poloze [20]. Jinak řečeno, pokud se může příznak objevit v jedné části obrazu, může se objevit kdekoli, a tedy i v jiné části obrazu.

2.4 Proces učení neuronové sítě

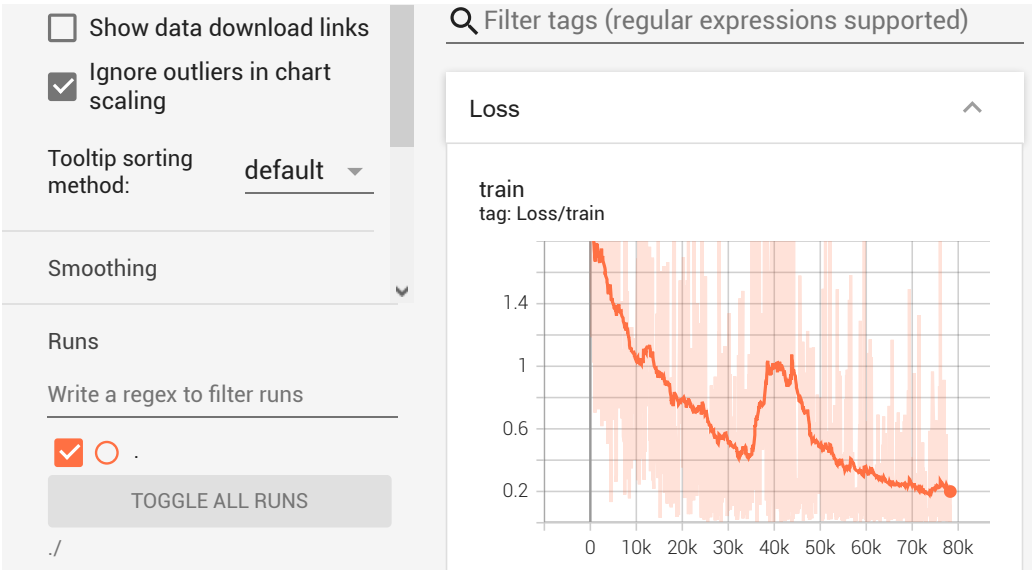
Účelem procesu je optimalizovat váhy neuronů (viz obrázek 2.1 vlevo) pro co nejlepší úspěšnost neuronové sítě. Dnes jediná použitelná metoda učení je metoda zpětného šíření chyby (backpropagation). Jedná se o učení s učitelem, kde se za učitele považuje v tomhle případě množina správně označených (anotovaných) snímků budov.

3 Dostupné optimalizované knihovny

Takových knihoven existuje celá řada. Většinou jsou jejich aplikační programátorské rozhraní dále jen API[1] přizpůsobeny hlavně pro programovací jazyk Python. Z nejznámějších bychom mohli vyjmenovat například TensorFlow nebo PyTorch.

3.1 Knihovna TensorFlow

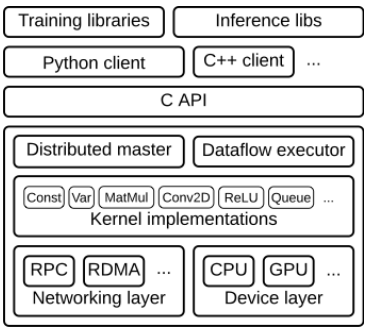
TensorFlow je open source[2] platforma pro strojové učení. Tuto open source knihovnu publikovala společnost Google v roce 2017 na platformu Github [45]. API pro jazyk Python se na učení konvolučních neuronových sítí používá její nadstavba knihovna Keras. Mezi užitečné nástroje knihovny patří takzvaný TensorBoard, který umí zobrazovat průběh učení neuronové sítě.



Obrázek 3.1: Nástroj TensorBoard

[1] Application Programming Interface (aplikační programátorské rozhraní)
[2] software s volně přístupnými zdrojovými kódy

Potenciál TensorFlow spočívá v přenositelnosti[3].



Obrázek 3.2: Architektura knihovny TensorFlow (obrázek převzat z [14])

Na obrázku 3.2 je vidět, že API definované v programovacím jazyku C odděluje uživatelský kód od kódu jádra knihovny, kde jsou implementovány algoritmy používané v uživatelském kódu. To znamená, že i když se používá verze pro Python a C++, tak se v budoucnu za minimálního úsilí bude dát použít i pro jiný programovací jazyk. Dále je samozřejmost podpora akcelerátoru, tudíž musí knihovna zajistit i efektivní paralelizaci. Více informací lze najít v [14].

3.1.1 Knihovna Keras

Keras, specializující se hlavně na hluboké neuronové sítě, slouží jako nástavba pro ostatní knihovny. Hlavně pro knihovnu TensorFlow. Cílem této nástavby je zjednodušit TensorFlow a usnadnit práci s přípravou vstupních dat pro učení neuronových sítí. Knihovna je psaná v drtivé většině jazykem Python.

3.2 Knihovna PyTorch

PyTorch je poměrně rozsáhlá knihovna s otevřeným kódem (open source), která je zaměřená na strojové učení. První verze byla vydána v roce 2016 společností Facebook [5]. Tato knihovna staví na základech další knihovny jménem Torch, která je popsána v následující sekci 3.2.1.

Jedna z výhod knihovny PyTorch je možnost efektivně spolupracovat s knihovnami, jako je například matematická knihovna NumPy.

Komponenty knihovny jsou navržené tak, aby šly jednoduše vylepšit či zaměnit. To znamená, že pokud někomu něco nevyhovuje, lze nahradit nevyhovující část za jinou, která je potřeba pro daný projekt.

PyTorch je psán z větší části programovacím jazykem C++, aby efektivně využíval výpočetní možnosti zařízení, na kterém je spuštěn. Tato skutečnost otevírá možnosti použití knihovny i v prostředích, kde se jazyk Python ne-

[3]software spustitelný na různých operačních systémech

hodí. Například pro operační systémy iOS a Android. Také jsou zde možnosti použití i v jiných programovacích jazycích, jako je třeba Java.

Další vlastností knihovny je přísné oddělení vykonávaného kódu ve skriptovacím jazyce Python od vlastních operací nad daty, které provádí jádro knihovny v C++ a následně operace rozdělí paralelně mezi procesory anebo akcelerátory. Toto je velice podobné schématu 3.2 v knihovně TensorFlow.

3.2.1 Torch

Torch je knihovna, která je napsána v programovacím jazyce C, kde je implementováno API (aplikační programátorské rozhraní) pro skriptovací jazyk Lua. První verze knihovny vyšla v roce 2002 [7], nicméně mezi její autory patří jména jako například Samy Bengio, což je bratr vědce, kterým je Yoshua Bengio, patřící mezi vědce úzce spojované s neuronovými sítěmi. Důležitou součástí knihovny jsou nástroje pro práci s vícerozměrnými daty. Další klíčovou funkcionalitou je podpora pro stavbu nejen neuronových sítí.

4 Výběr knihovny pro použití

Vybírat budeme mezi knihovnami PyTorch a TensorFlow. V roce 2022 jsou PyTorch i TensorFlow velmi vyspělé knihovny a jejich základní funkce pro práci s neuronovými sítěmi se výrazně překrývají [30].

PyTorch i TensorFlow nabízí známé architektury neuronových sítí jako ResNet či MobileNet. U obou knihoven je třeba před trénováním sestavit architekturu a hyper-parametry, jako je například ztrátová funkce[1].

Na druhou stranu, autoři v [6] narazili na nedostatek v knihovně TensorFlow při instalaci. Týkalo se to instalace platformy jménem CUDA pro spolupráci s akcelerátory, kde vyskytla nekompatibilita verzí knihovny TensorFlow a platformy CUDA, zatímco PyTorch, i když používá také platformu CUDA, si vybere potřebné ovladače z platformy CUDA a poskytuje je už v rámci knihovny. Dále popsali případ, kdy při nastavení hyper-parametry pro trénování architektury v TensorFlow nebyla v dané verzi podporována funkcionalita a bylo třeba vytvořit vlastní (podrobnosti lze najít v [6]). Zatím co PyTorch funkcionalitu podporoval.

Framework	*Throughput (higher is better)*					
	AlexNet	VGG-19	ResNet-50	MobileNet	GNMTv2	NCF
Chainer	778 ± 15	N/A	**219** ± 1	N/A	N/A	N/A
CNTK	845 ± 8	84 ± 3	210 ± 1	N/A	N/A	N/A
MXNet	**1554** ± 22	113 ± 1	218 ± 2	444 ± 2	N/A	N/A
PaddlePaddle	933 ± 123	112 ± 2	192 ± 4	**557** ± 24	N/A	N/A
TensorFlow	1422 ± 27	66 ± 2	200 ± 1	216 ± 15	9631 ± 1.3%	4.8e6 ± 2.9%
PyTorch	1547 ± 316	**119** ± 1	212 ± 2	463 ± 17	**15512** ± 4.8%	**5.4e6** ± 3.4%

Tabulka 4.1: Tabulka uvádějící, kolik obrázků za sekundu zpracují různé knihovny podle známých architektur neuronových sítí (obrázek převzat z [31])

Rychlost učení je důležitá. Z tabulky 4.1 je vidět, že pro jisté architektury neuronových sítí, které použijeme, je rychlejší knihovna PyTorch než TensorFlow. K podobným výsledkům přišli i rumunští autoři [6].

Je sice pravda, že rumunští autoři [6] testovali u architektur neuronových sítí i přesnost (accuracy[2]) a vyšlo, že TensorFlow je v tomto ohledu lepší, i tak převažují výhody knihovny PyTorch, proto ji budeme v této práci používat.

[1] Tato funkce popisuje funkci aktuální chybovosti sítě během učení, kterou je nutno minimalizovat.

[2] Metrika přesnosti se udává podílem správně klasifikovaných obrázků vůči všem klasifikovaným obrázkům (zjednodušeno).

5 Výběr architektonických slohů pro klasifikaci

V této kapitole shrneme výběr slohů, které po konzultaci s architektem budeme anebo nebudeme používat jako třídy pro klasifikaci. Architektonické slohy jsou detailněji popsány v kapitole IV.

- Starověký Egypt – Bohužel kvůli malému výskytu příkladů staveb se v seznamu tříd pro automatickou detekci neobjeví.
- Antika (Řecko, Řím) – Nebude vhodné zařazovat tento sloh do seznamu tříd pro automatickou detekci, kvůli velké podobnosti a návaznosti na architektonické styly, které přichází po Antice.
- Románský sloh – Románský sloh vzhledem početnosti staveb bude zahrnut do seznamu tříd pro automatickou detekci.
- Gotika – Gotika je vhodná pro automatickou detekci díky početnosti staveb, pokud nezahrneme Antiku, tak bude zahrnuta do seznamu tříd.
- Renesance – Výhodou je poměrně veliký počet staveb a pokud nezahrneme Antiku (Řecko, Řím) kvůli podobnosti, zařadíme Renesanci do seznamu tříd.
- Baroko – Vzhledem k početnosti barokních památek, bude tento sloh zařazen pro automatickou detekci.
- Secese – Díky své unikátnosti mezi ostatními zde zmíněnými slohy bude zahrnuta do seznamu tříd.
- Funkcionalismus – Tento sloh bude o zahrnut díky početnosti staveb.
- Kubismus – Tento specificky český sloh bude zahrnut pro automatickou detekci.
- Brutalismus – Tento typický jednoznačně rozpoznatelný sloh bude v seznamu tříd pro automatickou detekci.
- Islámská a arabská architektura – Tento architektonický sloh je vhodný pro automatickou detekci díky početnosti staveb a unikátnosti.

- Indická a khmérská architektura – Tato architektura je dobře rozpoznatelná. Měla by být součástí tříd pro automatickou detekci. Nicméně moc staveb v khmérském slohu nenajdeme, proto nebude zahrnuta do mezi slohy pro automatickou detekci.
- Slohy staré Číny, Japonska a Koreje – Tyto styly Východní Asie by se po konzultaci s architektem daly rozdělit do několika tříd. Nicméně kvůli jejich podobnosti je použití automatické detekce slohů v praxi takto vhodnější.

5.1 Shrnutí

Architektonické slohy pro klasifikaci	
Název slohu	Použijeme
Starověký Egypt	Ne
Antika (Řecko, Řím)	Ne
Románský sloh	Ano
Gotika	Ano
Renesance	Ano
Baroko	Ano
Secese	Ano
Funkcionalismus	Ano
Kubismus	Ano
Brutalismus	Ano
Islámská a arabská architektura	Ano
Indická a khmérská architektura	Ne*
Slohy staré Číny, Japonska a Koreje	Ano

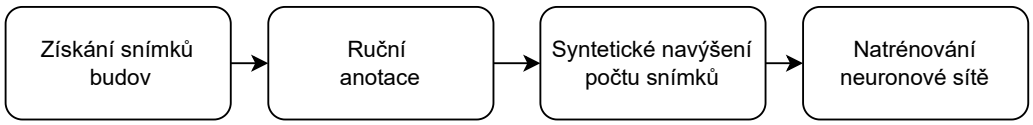
Tabulka 5.1: Výběr slohů

Poznámka

*Indickou a khmérskou architekturu nezařadíme z důvodu malého počtu dostupných obrázků v důsledku menšího počtu zachovaných staveb. Je sice připravená složka s touto třídou, ale natrénované architektury neuronových sítí ji až na výjimky nepodporují.

6 Návrh postupu přípravy dat

Cílem této bakalářské práce je vyvinout aplikaci pro klasifikaci budov. Postup je tedy následující. Nejprve je třeba získat velké množství obrázků budov, pak obrázky ručně anotovat (ručně zařadit každý obrázek do správné třídy), z ručně anotovaných snímků vytvořit množiny pro trénování konvolučních neuronových sítí, případně množiny synteticky rozšířit. Dále je třeba natrénovat architekturu neuronové sítě a následně ho použít pro klasifikaci ve vyvíjených aplikacích. Na obrázku 6.1 jsou přehledně popsány kroky potřebné k přípravě dat pro trénování.



Obrázek 6.1: Schéma postupu přípravy dat

6.1 Příprava dat

V následujících sekcích si popíšeme zvolené techniky přípravy tříd, které se použijí za vstup pro trénování neuronových sítí. Nejdříve se podíváme na získávání obrázků. K tomu je třeba si napsat program, který automaticky vyhledá a stáhne daný webový obsah, v našem případě obrázky budov (takzvaný crawler). Dále je třeba vytřídit vadné anebo nesouvisející snímky z programu crawler. Nakonec musíme získané třídy rozšířit, aby měly dostatečnou velikost. Toho dosáhneme různými úpravami kopií snímků.

6.2 Crawler

Nejprve bylo potřeba nalézt vhodný internetový vyhledávač. Nabízelo se jich mnoho, ale byl zvolen DuckDuckGo z následujících důvodů. Tento webový vyhledávač je známý svojí diskrétností. To znamená, že nesleduje své uživatele, ale co je nejdůležitější, má otevřené API. To je veliká výhoda oproti vyhledávači Google, který má sice lepší vyhledávací schopnosti, ale jeho API je placené.

Program byl napsán v jazyce Python. Byla využita knihovna DuckDuckGoImages z důvodu již implementované paralelizace pro větší rychlost stahování obrázků. Dále bylo třeba smazat obrázky, které byly buď poškozené

nebo neměly méně jak 24 bitů bitové hloubky (podpora méně barev v obrázku) či byly na libovolné straně menší než 320 pixelů. V následující tabulce 6.1 jsou shrnuty počty tříd po stažení crawlerem a validaci, zda nejsou poškozené.

Architektonické slohy po stažení	
Název slohu	Počet snímků
Románský sloh	737
Gotika	748
Renesance	767
Baroko	789
Secese	766
Funkcionalismus	655
Kubismus	647
Brutalismus	792
Islámská a arabská architektura	757
Indická a khmérská architektura	615
Slohy staré Číny, Japonska a Koreje	746

Tabulka 6.1: Počty snímků v třídách po stažení a smazání vadných položek

6.3 Manuální anotace

Na první pohled je sice možné přiřadit jakoukoliv budovu architektonickému slohu, pokud je daná budova postavená ryze v tom či onom slohu. To je ideální případ.

Bohužel v reálných případech může být na snímcích zachyceno více budov, když pak přidáme skutečnost, že jedna budova může být postavena ve více slozích v důsledku různých přestaveb.

Při manuální anotaci jsme zohlednili budovy, které na fotkách dominují, čímž tedy zabírají největší část plochy na snímcích. Dále bylo třeba smazat nesouvisející snímky. V tabulce 6.2 jsou shrnuty počty tříd po manuální anotaci.

Architektonické slohy po manuální anotaci		
Název slohu	Počet snímků	*%
Románský sloh	605	≈ 8,9
Gotika	675	≈ 9,9
Renesance	621	≈ 9,1
Baroko	646	≈ 9,5
Secese	687	≈ 10,2
Funkcionalismus	540	≈ 8,0
Kubismus	512	≈ 7,5
Brutalismus	731	≈ 10,9
Islámská a arabská architektura	691	≈ 10,2
Indická a khmérská architektura	445	≈ 6,6
Slohy staré Číny, Japonska a Koreje	623	≈ 9,2

Tabulka 6.2: Počty snímků v třídách po manuální anotaci *(podíl počtů jednotlivých tříd slohů vůči celkovému počtu obrázků v procentech)

6.4 Rozšíření množiny snímků pro trénování

Dále bylo třeba rozšířit trénovací množinu za účelem lepší úspěšnosti neuronových sítí. K tomu jsme použili knihovnu ImageMagick, která umožňuje hromadné zpracování obrázků. Dále jsme využili knihovnu Wand, která pracuje s knihovnou ImageMagick v programovacím jazyce Python. Pro zpracování snímků byly použity metody: zrcadlová projekce, multiplikativní šum, rozmazání pohybem, rotace o 10° na obě strany, výřezy 90% šířky a 90% výšky obrázku zleva a shora také 90% šířky a 90% výšky zprava a zdola. Kombinacemi rotací a výřezů s ostatními metodami bylo možné rozšířit třídy pro trénování až `28x` oproti původním počtům v tabulce 6.2.

6.4.1 Zrcadlová projekce

Zrcadlová projekce je jednoduchá projekce pixelů snímků v horizontálním směru. Jednoduše řečeno, levá strana snímku je na pravé straně a naopak. Na obrázku 6.2 je vidět příklad. Operace může pomoci například při klasifikaci budovy v různých časech (dopoledne a odpoledne), kdy může být budova nasvícena z jiného úhlu.


Obrázek 6.2: Vlevo: příklad zrcadlové projekce, vpravo: příklad multiplikativního šumu

6.4.2 Multiplikativní šum

Šum je způsoben zvlněním povrchu snímaných objektů nebo stíny, které vrhají složité objekty jako je listí, žaluzie nebo tmavé skvrny způsobené prachem v objektivu nebo ve snímači fotoaparátu. Při menší intenzitě viditelného světla se může projevit více. Příklad je vidět na obrázku 6.2 vpravo.

V této práci byla použita funkce `noise('multiplicative_gaussian')` z knihovny Wand.

Před přidáním šumu bylo třeba nejprve zpracovávaný obrázek částečně ztmavit a snížit barevnou sytost tak, aby upravený snímek odpovídal co nejvíce fotografiím pořízeným za šera. Parametry jasu byly sníženy ze 100% na 60%. Saturace byla snížena ze 100% na 35%.

6.4.3 Rozmazání pohybem

Jedná se o konvoluční operaci (popsána v sekci 2.4) se specifickou maticí jádra. V našem případě rozmazání pohybem bylo s maticí jádra o velikosti 5x5 pixelů.

Obrázek 6.3: Příklad rozmazaného snímku pohybem

6.5 Varianty trénovacích množin

Při rozšiřování trénovací množiny bylo třeba nejen vybrat správné kombinace úprav. Také bylo nutností vytvořit variantu s co nejlépe vyváženými

počty tříd. Bylo proto nutné v průběhu experimentů vyřadit třídu *Indická a khmérská architektura* kvůli malému počtu snímků.

6.5.1 Plně rozšířená množina

V této variantě jsme využili kombinace úprav obrázků (popsáno v sekci 6.4). Výsledkem byla množina obsahující 189 728 obrázků včetně třídy *Indická a khmérská architektura*. Bez zmíněné třídy množina obsahovala 177 268 obrázků.

I na nejmodernějším počítači, který byl k dispozici a jehož specifikace jsou uvedeny v sekci 7.2, trvala jedna trénovací a testovací fáze (epocha) okolo 2 hodin.

Vyvážená varianta

Pro vyvážení množiny se vzala z plně rozšířené varianty třída s nejmenším počtem obrázků a podle jejího počtu snímků se smazaly obrázky u ostatních tříd. Byla to, když nepočítáme indickou a khmérskou architekturu, třída Kubismus s 14 336 snímky, tím pádem měla pak výsledná množina 143 360 fotografií.

6.5.2 Množina bez rotací a ořezů (rozšířená množina)

Při tvorbě této varianty se využily pouze metody zrcadlové projekce, multiplikativního šumu a rozmazání pohybem, které rozšířily množinu tříd pro trénování přesně *čtyřikrát*. Celkem je to 27 104 obrázků i s indickou a khmérskou architekturou. Bez ní je v této variantě 25 324 obrázků.

S počtem 25 324 obrázků je učení neuronových sítí znatelně rychlejší, ale výsledné neuronové sítě budou nejspíše citlivější na mírné pootočení fotoaparátu nebo pokud na vstupní fotografii nebude budova úplně celá.

7 Architektury neuronových sítí knihovny PyTorch

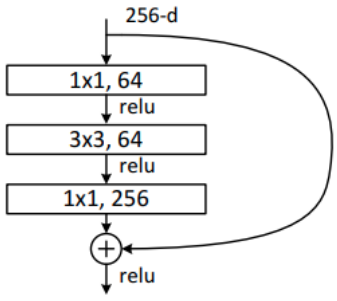
V kapitole se zaměříme na architektury, které byly použity pro učení na datových množinách popsaných v sekci 6.5. Pak se podíváme na jejich výsledky úspěšností, ze kterých se bude částečně odvíjet potenciál pro nasazení do mobilní nebo webové aplikace. Částečně proto, že budeme také hledět na paměťové nároky jednotlivých architektur.

7.1 Použité architektury

Zde stručně popíšeme typické prvky architektur. Všechny architektury byly použity z knihovny TorchVision, což je volitelná komponenta knihovny PyTorch [33]. Architektury jsou vybrané ze sekce určené pro strojové vidění a klasifikaci objektů na snímcích. Dále jsou inspirací z podobných experimentů jako [24], kde autor použil DenseNet121 s přesností 75.4% a ResNet50 s přesností 71.4%. Pro rychlejší naučení se použily již předtrénované verze architektur na (pro nás) neznámých obrázcích z knihovny TorchVision.

7.1.1 ResNet50

ResNet neboli Residual Network (česky zbytková síť) je typická svými zkratkovými spojeními (shortcut connections) v konvolučních vrstvách. V [15] jsou podrobně popsány vlastnosti této sítě. Zde byla použita architektura z knihovny TorchVision, nicméně existují i vylepšené architektury jako například ResNet50 ve verzi 1.5 od společnosti NVIDIA [29], což je upravená architektura sítě ResNet50 verze 1.0 od společnosti Microsoft [15]. Pro základní použití byla použita architektura ze zmíněné knihovny TorchVision.



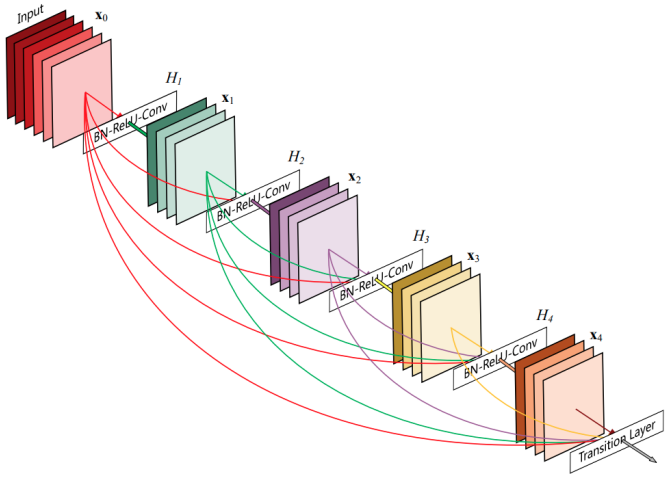
Obrázek 7.1: Zkratkové spojení (shortcut) konvolučních vrstev v architektuře ResNet50 (obrázek převzat z [15])

Paměťové nároky architektury ResNet50

Velikost architektury ResNet50 na pevném disku počítače zabírá zhruba 100 megabajtů. V tomto případě je architektura v porovnání s následujícími největší (kromě architektury ResNet152) a hodí se pro použití na stroji s lepší paměťovou kapacitou.

7.1.2 DenseNet121

Hustě propojená konvoluční síť neboli DenseNet je pojmenovaná po hustě propojených blocích (viz obrázek 7.2), kde každý vstup v rámci bloku se zkopíruje a jeho kopie jsou vstupem všech následujících konvolucí. Například na obrázku 7.2 je červený vstup zkopírován a vstupuje do všech následujících konvolucí v rámci bloku.



Obrázek 7.2: Příklad hustě propojeného bloku, který je součástí počátečních konvolučních vrstev (obrázek převzat z [16])

Takových bloků může být v konvolučních vrstvách více. Poslední z nich ústí do vrstevnaté neuronové části, viz obrázek 2.2. Podrobnější informace o vlastnostech této sítě můžeme najít v [16].

Paměťové nároky architektury DenseNet121

Velikost architektury DenseNet121 na pevném disku zabírá zhruba 30 megabajtů, proto lze použít i v mobilní aplikaci.

7.1.3 MobileNetV2

MobileNetV2 je optimalizovaná architektura od společnosti Google [39]. Její struktura je znatelně složitější než u předchozích výše popsaných sítí. Archi-

tektura této sítě obsahuje podobné prvky jako u architektury ResNet, kde jsou v konvolučních vrstvách také zkratková spojení (shortcut), která se na rozdíl od architektury ResNet redukují počet parametrů do dalších vrstev. Více informací lze naleznout v [39].

Paměťové nároky architektury MobileNetV2

Jedná se o paměťově optimalizovanou architekturu, která na pevném disku zabírá zhruba pouhých 14 megabajtů, proto ji lze použít i v přístrojích s paměťově omezenými možnostmi.

7.1.4 ResNet152

Síť ResNet152 patří do stejné skupiny jako ResNet50. Od sítě ResNet50 se liší nejen v počtu konvolučních vrstev, ale i v počtu vstupních parametrů. Podle experimentů v [15] by měla být ResNet152 až o 1% přesnější.

Paměťové nároky architektury ResNet152

Tuto síť je nevhodné ji použít v mobilní aplikaci, jelikož na pevném disku zabírá zhruba 236 megabajtů.

7.2 Výsledky učení

Výsledky jsou rozděleny do pěti částí. V každé části jsou na začátku popsána kritéria pro vyhodnocování. První část je uvedena s vyhodnocením podle odhadu pravděpodobnosti správné klasifikace, druhá je uvedena podle ztrátové funkce, třetí podle přesnosti (accuracy), čtvrtá podle F1 míry (F1 score) a v páté je pro přehlednost popsáno, ze kterého cyklu učení jsou prezentovány výsledky z prvních dvou částí a časy stráveným učením jednotlivých architektur neuronových sítí.

Data byla rozdělena na trénovací a validační množinu v poměru 4:1. Pro učení na datech plně vyvážené a plně rozšířené množiny byl vybrán notebook Lenovo Legion 5 s procesorem AMD Ryzen 7 a grafickou kartou NVIDIA GeForce RTX 3060. Pro učení na datech rozšířené množiny byl zvolen notebook HP Pavilion s procesorem Intel Core i7 a grafickou kartou NVIDIA GeForce GTX 960M.

Při učení neuronových sítí se u každé epochy[1] počítaly metriky z trénovací a validační fáze.

[1]Epocha je jeden cyklus, kdy proběhne trénovací a validační fáze.

7.2.1 Odhad pravděpodobnosti správné klasifikace

Odhad pravděpodobnosti správné klasifikace byl základním prvkem indikujícím úspěšnost. Byl spočten jako podíl správně klasifikovaných snímků (v rámci dané validační fáze) a všech snímků v dané validační fázi. V tabulce 7.1 jsou zobrazeny výsledky odhadů pravděpodobnosti správné klasifikace.

Výsledky odhadů pravděpodobnosti správné klasifikace (validační fáze)				
Vstupní data	ResNet50	DenseNet121	MobileNetV2	ResNet152
Plně rozšířená	78,56%	78,88%	75,59%	N/A%
Plně vyvážená	N/A%	76,03%	N/A%	N/A%
Rozšířená	80,70%	79,40%*	76,10%	72,44%

Tabulka 7.1: Výsledky odhadů pravděpodobnosti správné klasifikace jednotlivých naučených neuronových sítí (pro lepší vizualizaci převedeno na procenta, N/A = experiment nebyl prováděn)

Zde jsou uvedeny poznámky k tabulce 7.1. *DenseNet121 učící se s rozšířenou množinou obsahuje i třídu khmérské a indické architektury. U zmíněných výsledků se neobjevují data z trénovací fáze, které se pohybovaly u každé neuronové sítě při dosažení nejlepší hodnoty okolo 99%.

7.2.2 Ztrátová funkce

Dalším důležitým prvkem indikujícím úspěšnost byla ztrátová funkce. Jedná se o funkci chybovosti sítě během učení, kterou je nutno minimalizovat pro dosažení optimálního výsledku. Výsledky jsou zobrazeny v tabulce 7.2. Zde

Výsledky hodnot ze ztrátových funkcí (validační fáze)				
Vstupní data	ResNet50	DenseNet121	MobileNetV2	ResNet152
Plně rozšířená	1,4191	1,2977	1,6781	N/A
Plně vyvážená	N/A	1,8768	N/A	N/A
Rozšířená	0,9239	1,0550	1,2725	1,5878

Tabulka 7.2: Výsledky ztrátových funkcí jednotlivých naučených neuronových sítí (N/A = experiment nebyl prováděn)

opět nebudeme zveřejňovat celou tabulku z trénovací fáze, protože jsou hodnoty ztrátové funkce u všech podobné a pohybují se okolo 0,01 a menších.

7.2.3 Přesnost

Dalším prvkem pro vyhodnocování úspěšnosti experimentů byla zvolena přesnost (accuracy) podle vztahu, který lze najít v [26]. Testování přesnosti bylo prováděno až po naučení neuronových sítí. Výsledky jsou zobrazeny v tabulce 7.3. V tabulce 7.4 jsou takzvané `Top-3 accuracy`, což je přesnost, kdy se skutečná třída shoduje s některou ze 3 nejpravděpodobnějších tříd klasifikovaných neuronovou sítí.

Výsledky přesností (Top-1 accuracy)				
Naučeno z dat	ResNet50	DenseNet121	MobileNetV2	ResNet152
Plně rozšířená	0,783	0,779	N/A*	N/A
Plně vyvážená	N/A	0,756	N/A	N/A
Rozšířená	0,844	0,694	0,684	N/A

Tabulka 7.3: Výsledky přesností jednotlivých naučených neuronových sítí (N/A = experiment nebyl prováděn) testované na validační části z plně rozšířené množiny

Výsledky přesností (Top-3 accuracy)				
Naučeno z dat	ResNet50	DenseNet121	MobileNetV2	ResNet152
Plně rozšířená	0,921	0,92	N/A*	N/A
Plně vyvážená	N/A	0,91	N/A	N/A
Rozšířená	0,966	0,853	0,895	N/A

Tabulka 7.4: Výsledky přesností jednotlivých naučených neuronových sítí (N/A = experiment nebyl prováděn) testované na validační části z plně rozšířené množiny

7.2.4 F1 míra

Posledním prvkem pro vyhodnocování úspěšnosti experimentů byla zvolena F1 míra (F1 score). F1 míra byla spočtena stejně jako přesnost pomocí knihovny PyTorch Lightning [13]. Detailní informace týkající se F1 míry lze najít v [27]. Testování F1 míry bylo prováděno, stejně jako přesnost, až po naučení neuronových sítí. Výsledky jsou zobrazeny v tabulce 7.5 (výsledky této v tabulce se shodují s výsledky přesností (Top-1 accuracy) díky vyváženým klasifikačním třídám v neuronové síti).
Zde je uvedena poznámka k tabulkám 7.3, 7.4 a 7.5. *MobileNetV2, která se učila na plně rozšířené množině dat, nebylo možno otestovat z důvodu chyby při jejím uložení po natrénování.

Výsledky F1 míry				
Naučeno z dat	ResNet50	DenseNet121	MobileNetV2	ResNet152
Plně rozšířená	0,783	0,779	N/A*	N/A
Plně vyvážená	N/A	0,756	N/A	N/A
Rozšířená	0,844	0,694	0,684	N/A

Tabulka 7.5: Výsledky F1 míry jednotlivých naučených neuronových sítí (N/A = experiment nebyl prováděn) testované na validační části z plně rozšířené množiny

7.2.5 Nejúspěšnější epochy

Průběh učení pro každý neuronovou síť trval různě dlouho, od toho se odvíjel i různý počet epoch. V tabulce 7.6 z které epochy pocházejí nejlepší výsledky pro natrénované neuronové sítě přiložené k této práci. V tabulce 7.7 je uvedeno, jak dlouho trvalo naučení sítí se zadaným počtem epoch. Počet epoch se zadával jako hyper-parametr při spuštění skriptu pro učení neuronové sítě. Zde jsou uvedeny poznámky k tabulce 7.6. *Po sedmé epoše

Pořadová čísla epoch s nejlepšími výsledky / celkový počet epoch				
Vstupní data	ResNet50	DenseNet121	MobileNetV2	ResNet152
Plně rozšířená	12/29	10/14	19/49	N/A
Plně vyvážená	N/A	28/34	N/A	N/A
Rozšířená	12/14	9/11	65/100	5/15*

Tabulka 7.6: Pořadová čísla byla indexována od nuly a první číslo značí nejlepší epochu, druhé za lomítkem znamená celkový počet epoch (N/A = experiment nebyl prováděn)

Strávené hodiny učením jednotlivých neuronových sítí				
Vstupní data	ResNet50	DenseNet121	MobileNetV2	ResNet152
Plně rozšířená	47h/29	27h/14	28h/49	N/A
Plně vyvážená	N/A	85h/34	N/A	N/A
Rozšířená	27h/14	22h/11	50h/100	23h/7

Tabulka 7.7: Strávené hodiny učením jednotlivých neuronových sítí podle celkového počtu epoch ve formátu (čas v hodinách / počet epoch) (N/A = experiment nebyl prováděn)

bylo učení ResNet152 zastaveno, kvůli dlouhému trvání epoch a stabilním výsledkům z trénovacích fází, z čehož vyplynulo, že se neuronová síť už lépe nenaučí.

8 Implementace skriptů pro vytvoření natrénovaných neuronových sítí

V této kapitole si popíšeme implementaci skriptů pro přípravu dat a skriptů pro učení neuronových sítí, jejichž výsledkem jsou natrénované neuronové sítě.

8.1 Skripty pro přípravu dat

Tyto skripty jsou rozděleny do tří částí. První částí je crawler což program/skript, který automaticky stáhne obrázky či jiná média nebo text pomocí webového vyhledávače. Druhou částí jsou pomocné skripty pro kontrolu stažených obrázků, jejich přejmenování, třídění a statistické výpočty. Třetí částí je skript, který vytvoří data pro učení neuronových sítí a je z hlediska implementace nejzajímavější.

8.1.1 Použité technologie

Skripty byly napsány v programovacím jazyce Python, takže využívají základní knihovny, které tento jazyk nabízí. Mimo to jsou použity ale i technologie pro práci s obrázky.

DuckDuckGoImages [35]

Knihovna poskytující crawler, který byl popsán v sekci 6.2. Tento crawler se specializuje na získání obrázků ve formátu jpeg pomocí webového vyhledávače DuckDuckGo. V této práci byla vyžita knihovní metoda `download()`. Tato knihovna má velikou výhodu, a to že stahuje obrázky paralelně, tudíž stahování obrázků zabralo řádově minuty na každou třídu. Počty obrázků byly zveřejněny v tabulce 6.1.

ImageMagick [43]

ImageMagick je softwarový nástroj pro zpracování bitmapových obrázků. Byl vyvinut v programovacím jazyce C a v této práci použit ve spolupráci s knihovnou Wand.

Wand [23]

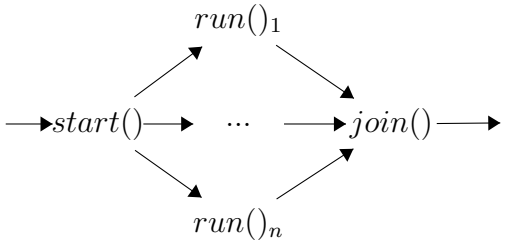
Knihovna pro jednoduché použití nástroje ImageMagick. Její výhodou byla jednoduchost, možnost použití v jazyce Python a poměrně přehledná dokumentace. Pro použití je třeba mít nainstalovaný ImageMagick.

8.1.2 Skript pro úpravu obrázků

Skript pro úpravu obrázků je skript pro paralelní zpracování již vytříděných snímků a jejich transformaci na data pro učení neuronových sítí. Skript se skládá z funkcí, které mají za úkol upravit snímek viz, 6.4. Tyto funkce volá procedura, která připraví kombinace upravených obrázků. Proceduru volá metoda `run()` ve třídě pojmenované `Edith`, která dědí od třídy vlákno a zpracovává tak jeden adresář obrázků dodaný parametrem konstruktoru. Ve spouštěcím skriptu je funkce `main()`, která načte parametry ze souboru `config.py` a vytvoří instance tříd `Edith` a následně nad nimi zavolá metodu `start()` v jednom cyklu a dalším cyklu metodu `join()`, což je znázorněno na následujících řádcích kódu.

```
1       for edith_class in edith_classes:
2           edith_class.start()
3
4       for edith_class in edith_classes:
5           edith_class.join()
```

Tento popsaný postup je aplikací návrhového vzoru farmer worker (farmář dělník), viz schéma 8.1. Skript kontroluje unikátnost názvů tříd ze souboru `config.py`, dá se považovat z pohledu paralelizace za bezpečný (thread safe).



Obrázek 8.1: Schéma návrhového vzoru farmář dělník aplikované v daném skriptu

8.2 Skripty pro učení neuronových sítí

Byly napsány a vyzkoušeny dva typy skriptů. Jeden čistě s knihovnou PyTorch a druhý s její nadstavbou, která se specializuje na maximální využití

výkonu počítače pro učení.

8.2.1 Použité technologie

Stejně jako předchozí skripty byly i tyto skripty psány v programovacím jazyku Python3. Zde jsou opět popsány využité knihovny, které nejsou součástí základní knihovny jazyka.

Dateutil [28]

Programovací jazyk Python sice nabízí základní knihovnu pro práci s jednotkami času, ale pohodlnější bylo využít tuto knihovnu. Byla použita její funkce `today()` za účelem vytvoření časové značky pro ukládané architektury neuronových sítí.

TorchVision [33]

Z knihovny TorchVision pocházejí použité neuronové sítě, ale TorchVision toho umí více, například umí vstupní obrázek přetransformovat do podoby vhodné ke vstupu do konvoluční neuronové sítě. Tento vstup se označuje jako tensor.

Torch [31]

Torch je základní část knihovny PyTorch, z knihovních metod byly kromě základních pro trénování použity matody jak pro spolupráci s akcelerátorem přes platformu CUDA, tak i pro statistiky, které se dají zobrazit pomocí nástroje TensorBoard, který je popsaný v sekci 3.1.

PyTorch Lightning [13]

PyTorch Lightning je nadstavba knihovny PyTorch. V této práci byla použita její knihovní třída jménem `LightningModule`, která reprezentuje v konstruktoru architekturu neuronové sítě a v metodách popisuje odděleně načtení dat pro trénování, validaci a další různé konfigurace. Pak stačí zdědit třídu `LightningModule` a překrýt její konstruktor a metody, vytvořit instanci této implementované třídy a třídy `Trainer` a na konec ve třídě `Trainer` zavolat metodu `fit(LightningModule)` s parametrem instance zděděné třídy.

8.2.2 Skript s PyTorch Lightning

Ve zděděné třídě byl překryt konstruktor, metody pro načítání trénovacích a validačních dat, pak následně metody definující trénovací a validační fáze.

Instance třídy `Trainer` byla volána s parametry, ze kterých jsou z hlediska výkonnosti nejdůležitější tyto:

- `devices="auto"` – Parametr, který automaticky nalezne akcelerátory (myšleno procesory, grafické karty a jiné) na základě následujícího parametru.
- `accelerator="gpu"` – Toto je typ zařízení, na kterém chceme učit neuronovou síť. V našem případě je to grafický akcelerátor.
- `strategy="dp"` – Strategie trénování, která popisuje paralelizaci s jedním akcelerátorem a procesorem.
- `benchmark=True` – Toto nastavení umožňuje zapnout vestavěné automatické ladění platformy CUDA a najít optimální algoritmus pro zpracovávání vstupních dat.
- `deterministic=True` – Použije deterministické[1] algoritmy v knihovně PyTorch.

8.2.3 Skript s knihovnou PyTorch

Skript používající pouze knihovnou PyTorch bez nástavby, přesněji balíček `torch` a balíček `torchvision`, se skládá z funkce `train_model()`, která iteruje podle počtu zadaných epoch. V každé iteraci se volá trénovací a následně validační fáze. Průběžné výsledky se zapisují do souboru, který lze monitorovat nástrojem TensorBoard (3.1). Parametry funkce `train_model()` jsou nastavení a architektura neuronové sítě. Kromě důležitých nastavení jako je použití platformy CUDA a povolení `benchmark` a `deterministic`, je velice důležitá hodnota `batch_size`, což je hodnota která dělí data na části pro paralelizaci načítání vstupních dat. Hodnota `batch_size` závisí na velikosti vstupních dat a výkonnosti počítače. V našem případě byla vhodná hodnota 4 a 5. Další součástí skriptu je třída `CnnPict` definující vstupní data a jejich konverzi na tensor popsaný v odstavci o TorchVision v sekci 8.2.1. V metodě `main()` se volí architektura neuronové sítě, nastavení platformy

[1]Deterministický je takový algoritmus, který má v každém svém kroku právě jednu možnost, jak pokračovat [25].

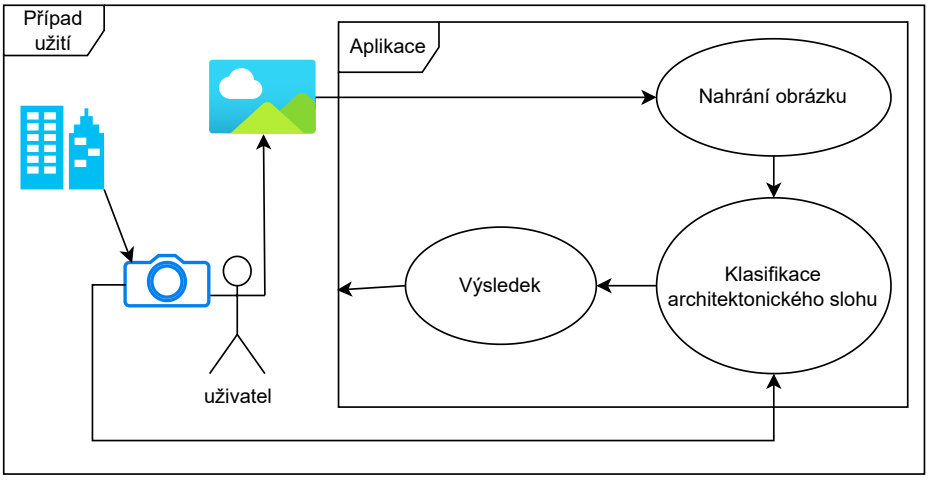
CUDA a další nastavení související s trénováním. Metoda `main()` načítá případné parametry příkazové řádky, a to je počet epoch učení se zmíněným parametrem `batch_size`. Při zadání nedostatečného počtu epoch je možné po zdárném ukončení pokračovat již s natrénovanou architekturou uloženou s koncovkou `_latest.pt`.

9 Implementované aplikace

Jedním z požadavků bylo naprogramovat aplikaci s jednoduchým uživatelským rozhraním. Aplikace by měla být mobilní nebo webová, proto si v této kapitole popíšeme nejen implementaci vyvíjených aplikací, ale i případy jejich užití.

9.1 Případ užití mobilní a webové aplikace

Případ užití je UML diagram popisující činnost aplikace z uživatelského hlediska. Samotný diagram je zobrazen na obrázku 9.1. Z diagramu je patrný postup, kdy se do aplikace nahraje obrázek přímo z knihovny obrázků anebo pomocí vestavěného fotoaparátu v zařízení (telefonu, tabletu, či jiných zařízení). Aplikace pak obrázek zpracuje a vrátí uživateli výsledek.



Obrázek 9.1: Případ užití pro mobilní a webovou aplikaci

9.2 Mobilní aplikace

Byla sice doporučena multiplatformní knihovna FireMonkey pro vývoj mobilní aplikace, ale nepodařilo se integrovat knihovnu PyTorch do vývojového prostředí C++ Builder [12]. Problém v načtení knihoven pro 64 bitovou architekturu, protože vyvíjená aplikace byla v průběhu vývoje primárně testována na tabletu s operačním systémem Android a 32 bitovou architekturou. Nicméně, jedná se o jednoduchou aplikaci, která načte obrázek a následně jej

zpracuje pomocí neuronové sítě a zobrazí výsledek, není tím pádem problém vytvořit aplikaci závislou na platformě, jak pro operační systém Android tak pro iOS, a tak byla mobilní aplikace omezena na platformu Android z důvodu překročení velikosti rozsahu této bakalářské práce.

Také bylo třeba i v knihovně PyTorch převést architekturu neuronové sítě do formátu vhodného pro mobilní zařízení, aby se dala použít v API knihovny pro jazyk Java.

9.2.1 Použité technologie

Aplikace byla psána ve vývojovém prostředí Android Studio a pro sestavení aplikace a automatické stažení knihoven byl vybrán nástroj Gradle. Dále byly použity níže zmíněné knihovny.

AndroidX [1]

AndroidX je knihovna, která například dokáže dodržet zpětnou kompatibilitu s nižšími verzemi Android. V této práci byla použita, protože vyvíjená aplikace byla v průběhu vývoje primárně testována na tabletu s operačním systémem Android ve verzi `5.0.1`.

PyTorch Android API [31]

Jedná se o API knihovny PyTorch, které umožňuje použít tuto knihovnu v programovacím jazyku Java. Existuje sice i API knihovny TorchVision (volitelná komponenta knihovny PyTorch), které bylo ze začátku použito, ale bylo neefektivní a zdržovalo klasifikátor běžící v aplikaci. Konkrétně se jednalo o převod obrázku do struktury pro vstup do neuronové sítě označované jako tensor, proto byl napsán vlastní kód, díky kterému nebylo vůbec potřeba API knihovny TorchVision.

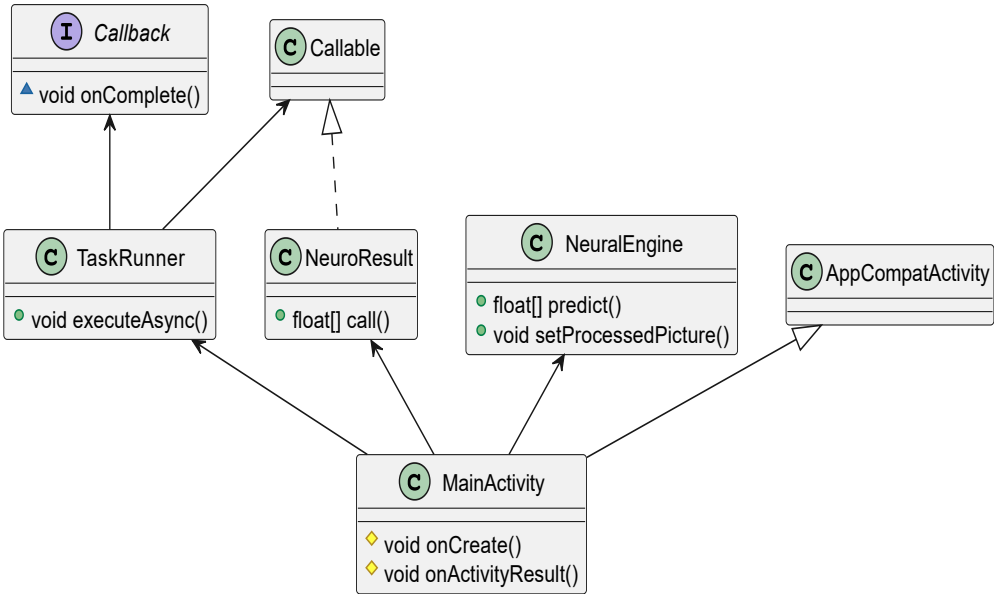
9.2.2 Struktura mobilní aplikace

Mobilní aplikace obsahuje jednu hlavní aktivitu aplikace, která komunikuje s uživatelem přes jednoduché uživatelské rozhraní znázorněné na obrázku I.2, které obsahuje tlačítka pro zobrazení informací o aplikaci, nahrání snímku z knihovny obrázků a nahrání obrázku z fotoaparátu. Třída `MainActivity` (popisující aktivitu aplikace) dědí od třídy `AppCompatActivity`, kterou můžeme nalézt v balíčku knihovny AndroidX.

Dále je zde třída `NeuralEngine`, která má na starost samotnou klasifikaci obrázku. Při startu aplikace se vytvoří instance třídy `NeuralEngine`, která

si při vytváření instance načte do paměti neuronovou síť uloženou s příponou `pt`. Pokud dojde k chybě při načítání neuronové sítě, aplikace vypíše chybovou hlášku a skončí.

Při načtení obrázku po obsluze tlačítka pro načtení nebo pro vyfocení se zavolá metoda `predict()` třídy `NeuralEngine`, která zpracuje obrázek do již zmíněného tensoru a následně tensor klasifikuje. Klasifikace obrázku probíhá asynchronně pomocí tříd z balíčku `java.util.concurrent`. UML diagram tříd je zobrazen na obrázku 9.2.



Obrázek 9.2: UML diagram tříd aplikace

9.2.3 Problémy při vývoji aplikace pro Android

První verze aplikace nepoužívala balík `java.util.concurrent`, ale třídu `AsyncTask` z balíčku `android.os`, která je označována jako zastaralá a Android ve verzi 12 třídu `AsyncTask` již nepodporuje.

9.2.4 Scénářové testování

Aplikace se testovala jak se zapnutými tak i vypnutými právy pro čtení a zápis do souborového systému. Bez přidělených práv se kontrolovalo, zda aplikace vypíše patřičné chybové hlášky. V tabulce 9.1 jsou vidět výsledky testů na různých zařízeních. Záznamy se značkou hvězdičky (*) jsou virtuální zařízení testované v Android Studiu. Aplikace byla otestována manuálně

díky jednoduchému uživatelskému rozhraní. Testovací scénáře jsou podrobně popsány v sekci II.

Název zařízení	Android verze	Funkční
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

Tabulka 9.1: Testy aplikace při načtení i vyfotografování obrázku, (*) testované na virtuálním zařízení

9.3 Webová aplikace

Pro implementaci webové aplikace se nabízelo několik postupů. Jedním z nich byl návod přímo na webových stránkách knihovny PyTorch [38], kde autor návodu popisoval, jak vytvořit webový server pomocí knihovny Flask v jazyce Python. Výhodou tohoto přístupu bylo, že se architektura neuronové sítě nemusela konvertovat do jiného formátu a daly se použít postupy pro běh neuronové sítě přímo v knihovně PyTorch.

Na druhou stranu se vyskytly problémy s velikostí knihovny PyTorch při vytváření spustitelného souboru nástrojem PyInstaller. Tento problém byl vyřešen použitím knihovny ONNX Runtime, která je popsána v sekci 9.3.1. Dalším problémem byla samotná knihovna Flask, pomocí které se sice dá velice rychle napsat webová aplikace, ale při spuštění podle výše uvedeného manuálu byla zobrazena varovná hláška sdělující, že není vhodné použít server zabudovaný v knihovně pro použití v běžném provozu. Webová aplikace je sice implementována výše uvedeným postupem, ale nakonec bylo rozhodnuto přepsat aplikaci do programovacího jazyka Javy přesněji do frameworku[1] Spring.

9.3.1 Použité technologie

Aplikace byla psána ve vývojovém prostředí IntelliJ IDEA a pro sestavení aplikace byl vybrán nástroj Gradle. Dále byly použity níže zmíněné knihovny nebo programovací jazyky.

[1]Framework je software, který poskytuje například soubor knihoven v dané problematice nebo také může předpisovat strukturu implementované aplikace.

Spring Boot [42]

Spring Boot je součást frameworku Spring, která se specializuje na aplikace sestavovány nástrojem Gradle. Jedná se o podobný přístup jako výše popsaná knihovna Flask, ale její výhodou je komplexnost konfigurací webové aplikace. V této práci byly konfigurace využity například pro kontrolování velikosti nahrávaných souborů v souboru `application.properties`. Dále bylo využito nástroje Thymeleaf pro tvorbu šablon webových stránek, který je plně integrován do frameworku Spring.

Bootstrap [3]

Bootstrap je sada šablon obsahující již připravené kaskádové styly pro vzhled webové stránky. V této práci byl použit za účelem vylepšení vzhledu aplikace, a také pro optimalizaci vzhledu na mobilním telefonu či tabletu.

ONNX Runtime [9]

ONNX Runtime je knihovna pro urychlení běhu neuronových sítí od firmy Microsoft.

Pro použití jakékoliv architektury neuronové sítě v programovacím jazyku Python bylo vhodné využít konverzi do formátu ONNX, což nejen zrychlilo běh aplikací ale i nároky na paměťové prostředky stroje, na kterém aplikace běží. Při distribuci aplikace nebylo potřeba zahrnovat celou knihovnu PyTorch, která má řádově jednotky GB[2], ale stačilo použít knihovnu ONNX Runtime, které stačí řádově desítky MB[3].

Aktuálně tuto knihovnu používá webová aplikace jak v jazyce Python, tak webová aplikace psaná v jazyce Java (zde bylo vhodnější použít ONNX Runtime, kvůli snadnější dostupnosti pomocí Gradle, což je nástroj pro automatické sestavení programu s knihovnami).

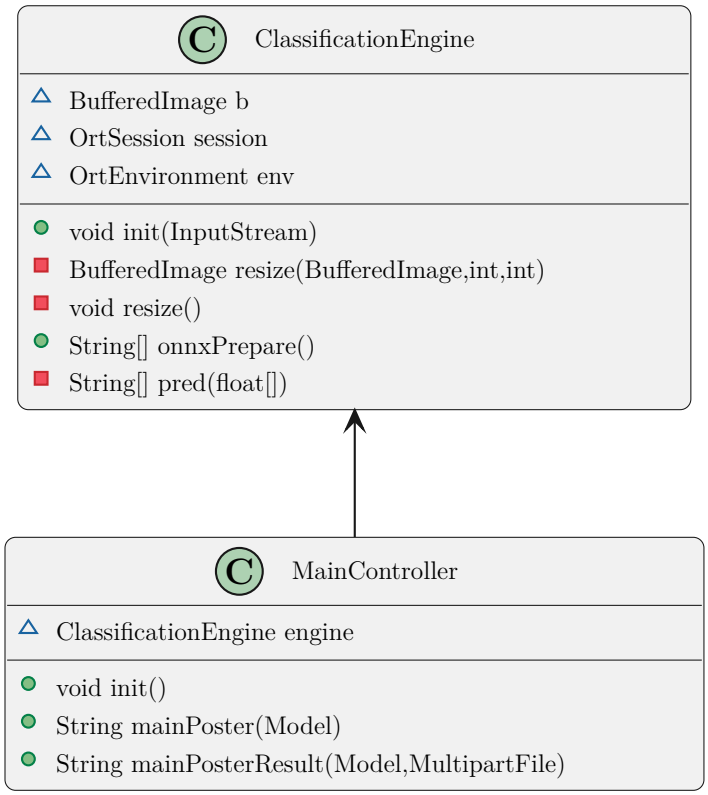
JavaScript

Skriptovací jazyk JavaScript byl zde použit pro rychlé zobrazování obrázků ve webovém prohlížeči.

[2]GiB = gigabyte
[3]MiB = megabyte

9.3.2 Struktura webové aplikace

Knihovna Spring Boot již částečně předepisuje strukturu implementované aplikace, a tou je architektura MVC[4]. V architektuře MVC je aplikace rozdělená podle aplikační logiky do tří částí. V aplikační logice model reprezentuje logiku klasifikace do architektonického slohu načteného obrázku, ovladač (controller) zpracovává požadavky od uživatele a nakonec pohled zobrazuje uživateli výsledná data ve formě webové stránky pomocí šablon nástroje Thymeleaf.



Obrázek 9.3: UML diagram tříd webové aplikace

9.3.3 Scénářové testování

Webová aplikace byla testována stejně jako mobilní aplikace scénářovým testováním. Testovací scénáře lze najít v sekci III. V tabulkách 9.2, 9.3 a 9.4 jsou shrnuty výsledky testování.

[4]Model, View, Controller (Model pracující s daty aplikace/databází, View zobrazující data uživateli, Controller zpracovávající požadavky od uživatele a řídící View a Model), což česky znamená model, pohled, ovladač.

Operační systém	Webový prohlížeč	Verze	Funkční
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO*
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO*
Windows 11	Opera	v85.0.4341.60	ANO
iOS	Firefox	v99.0	ANO
iOS	Google Chrome	v100.0.4896.85	ANO
iOS	Safari	v15.4.1	ANO
Android	Google Chrome	v43.0.2357.93	ANO

Tabulka 9.2: Testy webové aplikace při načtení obrázku se správně zobrazeným výsledkem

Operační systém	Webový prohlížeč	Verze	Funkční
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO*
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO*
Windows 11	Opera	v85.0.4341.60	ANO

Tabulka 9.3: Testy webové aplikace bez povoleného JavaScriptu

Operační systém	Webový prohlížeč	Verze	Funkční
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO*
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO*
Windows 11	Opera	v85.0.4341.60	ANO

Tabulka 9.4: Testy webové aplikace s blokovanými Cookies

Poznámky

V testovaných webových prohlížečích se až na výjimky záměrně neobjevuje prohlížeč Google Chrome (na Windows 11, Linux, iOS) ani prohlížeč Edge (na Windows 11) ani Opera (na Windows 10) z důvodu stejného jádra (browser engine) jménem Chromium, které také používá prohlížeč Opera. *V prohlížeči Firefox se vyskytla chyba při načítání frameworku Bootstrap. Při

základním nastavení se Bootstrap nenačetl, protože se načítal přes externí adresu pomocí jazyka JavaScript. Po dokončení testování byla chyba opravena použitím načtení přímo z webové aplikace a testování proběhlo znovu už bez chyby.

9.4 Verzování

Jednalo o rozsáhlejší práci, proto bylo vhodné práci rozdělit do projektů. V prvním projektu se řešila příprava dat a učení neuronových sítí, v druhém webová aplikace v jazyku Python, ve třetím webová aplikace s aplikačním rámcem Spring a ve čtvrtém mobilní aplikace. Všechny projekty se od začátku verzovaly pomocí nástroje Git s úložištěm GitLab.

9.4.1 Git

Git je nástroj pro správu verzí. Od jiných nástrojů spravujících zdrojový kód například SVN se liší tím, že je distribuovaný. To znamená, že nemusí mít jedno centrální úložiště se serverem, který řídí celé verzování s architekturou klient server.

9.4.2 GitLab CI/CD

GitLab neumožňuje pouze ukládání změn z nástroje Git, ale podporuje například i službu sestavování zdrojového kódu do spustitelných aplikací. Tato služba je označována jako CI (Continuous Integration) a v této práci byla využita. GitLab dále poskytuje i automatickou distribuci spustitelných aplikací přímo až do produkčního prostředí. Tato služba se nazývá Continuous Deployment (CD), ale nebyla použita v této práci.

10 Závěr

Na základě znalostí získaných při studiu zaměřeném na klasifikaci snímků konvolučními neuronovými sítěmi a prozkoumáním možností dostupných optimalizovaných knihoven byly navrhnuty a vypracovány postupy přípravy trénovacích dat klasifikátoru. Proběhla také konzultace s architektem, kdy byly probrané architektonické slohy s ohledem na vhodný návrh klasifikačních tříd. Byla naprogramována mobilní aplikace určená pro operační systém Android a webová aplikace. Obě aplikace mají jednoduché uživatelské rozhraní. Po nahrání anebo vyfocení snímku provede neuronová síť, která je implementována v rámci každé aplikace klasifikaci s cílem určit, o jaký architektonický sloh se jedná.

Implementační postupy v rámci každé aplikace bylo nutné v průběhu měnit. V mobilní aplikaci, která byla pro operační systém Android, bylo nutné vyměnit třídu `AsyncTask`, která je označena jako zastaralá (více informací lze najít v sekci 9.2.3), za třídy z balíčku `java.util.concurrent`. Webová aplikace, která byla původně vyvíjena v programovacím jazyce Python, byla nakonec psána v programovacím jazyce Java.

Obě aplikace byly testovány scénářovým testováním, které bylo navrženo v zadání. Scénářové testování po opravení chyby v načítání aplikačního rámce Bootstrap u webové aplikace proběhlo úspěšně u obou aplikací.

Přesnost klasifikace se pohybovala v rozmezí 68 až 84 procent, což je s ohledem na komplexnost řešeného problému velice uspokojivé. Detailně je přesnost popsána v sekci 7.2.

U aplikací se může stát, že při klasifikaci dvou snímků s podobným obsahem se mohou výsledky lišit. Klasifikátor také neumí rozlišit, zda na nahraném snímku je anebo není budova. Řešením by bylo přidat do aplikace rozpoznávač.

V průběhu implementace mobilní aplikace byl odhalen nedostatek, který zpomaloval její běh. Problém byl v knihovně TorchVision, kde byla použita výpočetně složitá implementace. Řešením bylo napsat vlastní kód konverze vstupních obrázků pro vstup do neuronové sítě.

Výhodou mobilní aplikace je, že není třeba se připojovat k žádnému zařízení a klasifikace se vykoná přímo na daném mobilním zařízení. Aplikace také podporuje kromě angličtiny i češtinu a tmavý i světlý režim. U webové aplikace je výhodou responzivní design, který se přizpůsobí například tabletu či mobilnímu telefonu. Aplikace by se mohly používat při výuce architektonických slohů ve škole nebo jen tak pro zábavu.

Literatura

[1] *AndroidX* [online]. 2022. [cit. 17. dubna 2022]. Dostupné z: https://mvnrepository.com/artifact/androidx.appcompat/appcompat/1.4.1.

[2] Bochenek, G. *Notre-Dam* [online]. en.wikipedia.org, 2011. [cit. 18. února 2022]. Dostupné z: https://en.wikipedia.org/wiki/Notre-Dame_de_Paris#/media/File:Notre-Dame_de_Paris-France.JPG.

[3] Bootstrap Core Team. *Bootstrap* [online]. 2021. [cit. 20. dubna 2022]. Dostupné z: https://github.com/twbs/bootstrap/tree/v4.6.1.

[4] [online]. kudyznudy.cz, 2016. [cit. 22. února 2022]. Dostupné z: https://www.kudyznudy.cz/ceska-nej/architektonicke/brutalismus-v-cr-architektura-drsneho-betonu.

[5] Chintala, S. [online]. 2016. [cit. 1. března 2022]. Dostupné z: https://github.com/pytorch/pytorch/releases/tag/v0.1.1.

[6] Chirodea et al. *Comparison of Tensorflow and PyTorch in Convolutional Neural Network - based Applications*. IEEE, 2021. doi: 10.1109/ECAI52376.2021.9515098. ISBN 978-1-6654-2534-6.

[7] Collobert, R. – Bengio, S. – Mariéthoz, J. *Torch: A Modular Machine Learning Software Library* [online]. 2002. [cit. 1. března 2022]. Dostupné z: https://publications.idiap.ch/downloads/reports/2002/rr02-46.pdf.

[8] [online]. wikimedia.org, 2006. [cit. 28. února 2022]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c5/Konvoluce_2rozm_diskretni.jpg/1280px-Konvoluce_2rozm_diskretni.jpg.

[9] developers, O. R. *ONNX Runtime* [online]. 2018. [cit. 19. dubna 2022]. Dostupné z: https://github.com/microsoft/onnxruntime.

[10] *Dům u Černé matky Boží* [online]. novinky.cz, 2015. [cit. 20. února 2022]. Dostupné z: https://www.novinky.cz/kultura/clanek/dum-u-cerne-matky-bozi-opet-prezentuje-cesky-kubismus-334031.

[11] [online]. hisour.com, 2017. [cit. 20. února 2022]. Dostupné z: https://www.hisour.com/khmer-architecture-31146/.

[12] Embarcadero. *C++Builder* [online]. 2021. [cit. 12. dubna 2022]. Dostupné z: https://www.embarcadero.com/products/cbuilder/starter.

[13] Falcon, W. – The PyTorch Lightning team. *PyTorch Lightning* [online]. 2019. [cit. 11. dubna 2022]. Dostupné z: https://github.com/PyTorchLightning/pytorch-lightning.

[14] Hathidara, A. [online]. github.com/tensorflow/, 2020. [cit. 22. února 2022]. Dostupné z: https://github.com/tensorflow/docs/blob/master/site/en/r1/guide/extend/architecture.md.

[15] He, K. et al. *Deep Residual Learning for Image Recognition* [online]. 2015. [cit. 8. března 2022]. Dostupné z: http://arxiv.org/abs/1512.03385.

[16] Huang, G. – Liu, Z. – Weinberger, K. Q. *Densely Connected Convolutional Networks* [online]. 2016. [cit. 8. března 2022]. Dostupné z: http://arxiv.org/abs/1608.06993.

[17] HUMUSOFT s.r.o. [online]. technickytydenik.vshcdn.net, 2017. [cit. 18. února 2022]. Dostupné z: https://technickytydenik.vshcdn.net/obrazek/5a12cd6643773/12.jpg.

[18] Jarvis, D. *Abu Simbel* [online]. worldhistory.org, 2013. [cit. 16. února 2022]. Dostupné z: https://www.worldhistory.org/image/1007/the-small-temple-abu-simbel/.

[19] Jarvis, D. *Pantheon* [online]. worldhistory.org, 2013. [cit. 16. února 2022]. Dostupné z: https://www.worldhistory.org/image/1272/pantheon-rome/.

[20] LeCun, Y. – Bengio, Y. – Hinton, G. *Deep learning* [online]. cs.toronto.edu, 2015. [cit. 12. února 2022]. Dostupné z: https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf.

[21] Mark, J. J. *Abu Simbel* [online]. worldhistory.org, 2018. [cit. 9. února 2022]. Dostupné z: https://www.worldhistory.org/Abu_Simbel/.

[22] Mařík, V. – Stěpánková, O. – Lažanský, J. *Umělá inteligence*. Academia, 1993. ISBN 80-200-0496-3.

[23] McConville, E. *Wand* [online]. 2021. [cit. 10. dubna 2022]. Dostupné z: https://github.com/emcconville/wand.

[24] Mo, W. *architecture-style-recognizer* [online]. https://github.com, 2020. [cit. 24. března 2022]. Dostupné z: https://github.com/warrenmo/architecture-style-recognizer.

[25] Neckář, J. *algoritmy.net* [online]. 2016. [cit. 12. dubna 2022]. Dostupné z: https://www.algoritmy.net.

[26] Nicki Skafte Detlefsen et al. *TorchMetrics - Measuring Reproducibility in PyTorch - Accuracy* [online]. 2022. [cit. 3. května 2022]. Dostupné z: https://torchmetrics.readthedocs.io/en/stable/classification/accuracy.html.

[27] Nicki Skafte Detlefsen et al. *TorchMetrics - Measuring Reproducibility in PyTorch - F1 Score* [online]. 2022. [cit. 3. května 2022]. Dostupné z: https://torchmetrics.readthedocs.io/en/stable/classification/f1_score.html.

[28] Niemeyer, G. *dateutil* [online]. 2021. [cit. 11. dubna 2022]. Dostupné z: https://github.com/dateutil/dateutil.

[29] NVIDIA. *resnet50v1.5* [online]. pytorch.org, 2021. [cit. 18. března 2022]. Dostupné z: https://pytorch.org/hub/nvidia_deeplearningexamples_resnet50/.

[30] O'Connor, R. *PyTorch vs TensorFlow in 2022* [online]. 2021. [cit. 5. března 2022]. Dostupné z: https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2022/.

[31] Paszke, A. et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library* [online]. Curran Associates, Inc., 2019. [cit. 15. února 2022]. Dostupné z: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[32] [online]. developers.google.com, 2021. [cit. 28. února 2022]. Dostupné z: https://developers.google.com/machine-learning/practica/image-classification/images/maxpool_animation.gif.

[33] pytorch. *torchvision* [online]. pytorch.org, 2017. [cit. 10. dubna 2022]. Dostupné z: https://github.com/pytorch/vision.

[34] [online]. pametnaroda.cz, 2019. [cit. 18. února 2022]. Dostupné z: https://www.pametnaroda.cz/sites/default/files/styles/article_small/public/Rathaus-pilsen_crop(1).jpg.

[35] Riquelme, J. E. P. *DuckDuckGoImages* [online]. github.com, 2021. [cit. 10. dubna 2022]. Dostupné z: https://github.com/JorgePoblete/DuckDuckGoImages.

[36] *Perceptron* [online]. researchgate.net, 2013. [cit. 10. února 2022]. Dostupné z: https://www.researchgate.net/figure/A-Single-Perceptron-Unit_fig1_341709718.

[37] [online]. britannica.com, 2019. [cit. 18. února 2022]. Dostupné z: https://www.kudyznudy.cz/aktivity/rotunda-sv-jiri-a-sv-vojtecha-na-ripu.

[38] Sajjanshetty, A. *Deploying PyTorch in Python via a REST API with Flask* [online]. pytorch.org, 2021. [cit. 19. dubna 2022]. Dostupné z: https://pytorch.org/tutorials/intermediate/flask_rest_api_tutorial.html.

[39] Sandler, M. – Howard, A. G. – Zhu, M. *Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation* [online]. 2018. [cit. 8. března 2022]. Dostupné z: http://arxiv.org/abs/1801.04381.

[40] [online]. vytvarna-vychova.cz, 2015. [cit. 22. února 2022]. Dostupné z: https://vytvarna-vychova.cz/secese/.

[41] *Granada* [online]. whereandwander.com, 2013. [cit. 14. února 2022]. Dostupné z: http://www.whereandwander.com/wp-content/uploads/2013/12/Granada-Spain-Alhambra-Generalife-Gardens-3.jpg.

[42] SpringSource. *Spring Boot* [online]. 2022. [cit. 19. dubna 2022]. Dostupné z: https://github.com/spring-projects/spring-boot.

[43] The ImageMagick Development Team. *ImageMagick* [online]. 2021. [cit. 10. dubna 2022]. Dostupné z: https://imagemagick.org.

[44] Ulak, J. [online]. britannica.com, 2019. [cit. 15. února 2022]. Dostupné z: https://cdn.britannica.com/02/210202-050-D644C84B/Horyu-ji-Temple-Ikaruga-Nara-Japan-Buddhism.jpg.

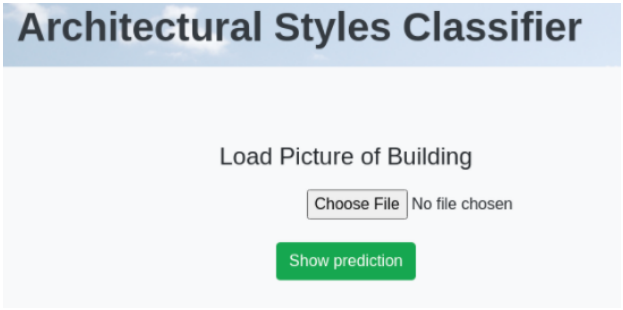
[45] Vasudevan, V. – Irving, G. – Feng, Y. *TensorFlow, 1.0.0* [online]. 2017. [cit. 20. února 2022]. Dostupné z: https://github.com/tensorflow/tensorflow/blob/07bb8ea2379bd459832b23951fb20ec47f3fdbd4/RELEASE.md.

[46] [online]. 2019. [cit. 18. února 2022]. Dostupné z: https://www.ngprague.cz/o-nas/budovy/veletrzni-palac.

I Uživatelská dokumentace

I.1 Webová aplikace

Ke spuštění je třeba mít zapnutý JavaScript a neblokovat cookies webové stránky. Pak stačí načíst obrázek kliknutím na tlačítko `Choose File` zobrazené na obrázku I.1.

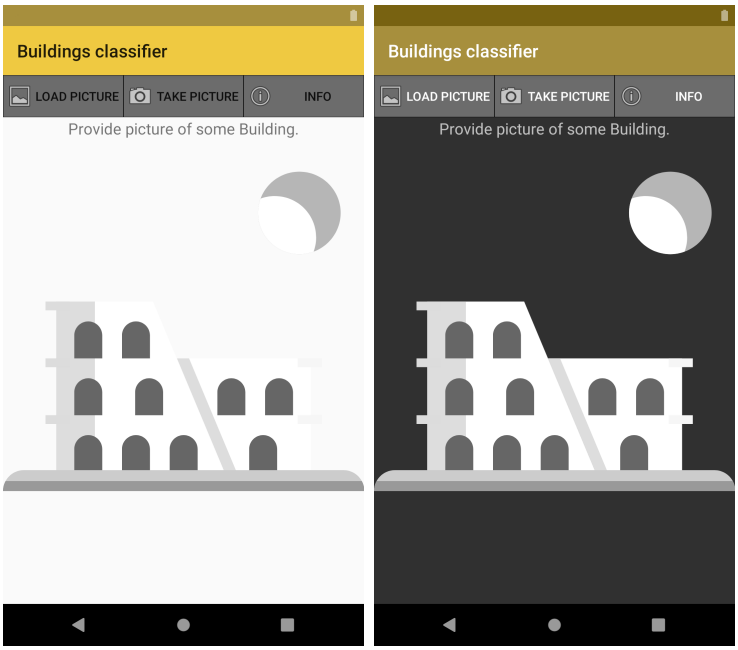


Obrázek I.1: Výřez úvodní obrazovky

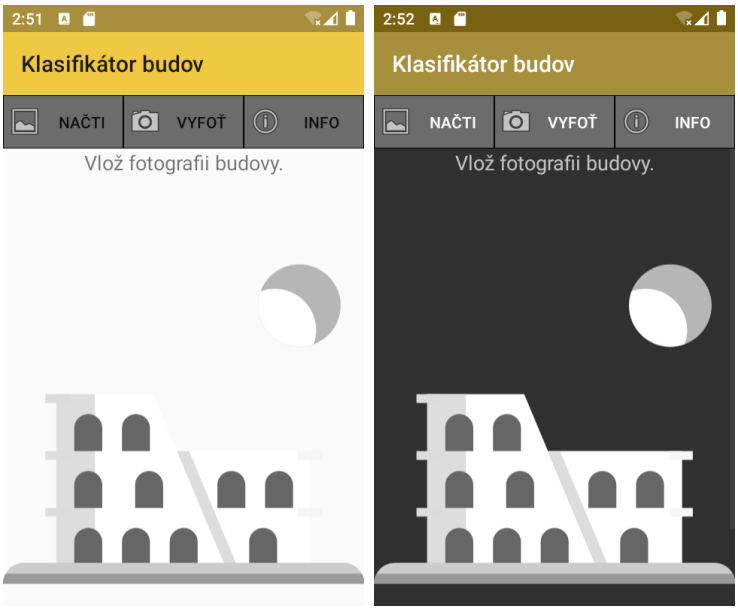
I.2 Mobilní aplikace

Mobilní aplikace obsahuje tři tlačítka, viz obrázky I.2 a I.3. První tlačítko načte snímek z galerie obrázků, druhé tlačítko otevře fotoaparát a následně nahraje snímek do aplikace, a třetí zobrazí informace o aplikaci. Po načtení obrázku se zobrazí výsledek. Zobrazení podporuje posouvání obsahu (scrollbar).

V této kapitole jsou popsány testovací scénáře a následně i testovací reporty.



Obrázek I.2: Uživatelské rozhraní pro aplikaci Android



Obrázek I.3: Uživatelské rozhraní pro aplikaci Android (čeština)

II Scénářové testování mobilní aplikace

II.1 Testovací scénáře

II.1.1 První scénář

1. Nainstalovat aplikaci. Požadovala aplikace jiná uživatelská oprávnění kromě přístupu k fotkám, souborům a médiím?
2. Udělit oprávnění. Dostala Aplikace požadované oprávnění?
3. Spustit aplikaci. Vykreslila se obrazovka podle vzoru jednoho z obrázků I.2 a I.3?
4. Zvolit tlačítko s ikonou fotoaparátu. Otevřel se fotoaparát?
5. Vyfotografovat a počkat na výsledky. Zobrazily se výsledky?
6. Zvolit tlačítko s ikonou rámečku. Otevřela se galerie obrázků?
7. Zvolit obrázek. Načetl se vybraný obrázek a jsou vzápětí vidět i výsledky?

II.1.2 Druhý scénář

1. Nainstalovat aplikaci. Požadovala aplikace jiná uživatelská oprávnění kromě přístupu k fotkám, souborům a médiím?
2. Neudělit oprávnění. Dostala Aplikace požadované oprávnění?
3. Spustit aplikaci. Vykreslila se obrazovka podle vzoru jednoho z obrázků I.2 a I.3?
4. Zvolit tlačítko s ikonou fotoaparátu. Otevřel se fotoaparát?
5. Vyfotografovat a počkat na výsledky. Zobrazily se výsledky?
6. Zvolit tlačítko s ikonou rámečku. Otevřela se galerie obrázků?
7. Zvolit obrázek. Načetl se vybraný obrázek a jsou vzápětí vidět i výsledky?
8. Zobrazil se dialog s chybovou hláškou?

II.2 Testovací report

První scénář

1. Nainstalovat aplikaci. Požadovala aplikace jiná uživatelská oprávnění kromě přístupu k fotkám, souborům a médiím?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	NE
Prestigio PMT3777 3	v5.1	NE
Galaxy Nexus (*)	v7.1.1	NE
Pixel 2 (*)	v10.0	NE
Nexus S (*)	v11.0	NE

2. Udělit oprávnění. Dostala Aplikace požadované oprávnění?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

3. Spustit aplikaci. Vykreslila se obrazovka podle vzoru jednoho z obrázků I.2 a I.3?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

4. Zvolit tlačítko s ikonou fotoaparátu. Otevřel se fotoaparát?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

5. Vyfotografovat a počkat na výsledky. Zobrazily se výsledky?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

6. Zvolit tlačítko s ikonou rámečku. Otevřela se galerie obrázků?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

7. Zvolit obrázek. Načetl se vybraný obrázek a jsou vzápětí vidět i výsledky?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

Druhý scénář

---

1. Nainstalovat aplikaci. Požadovala aplikace jiná uživatelská oprávnění kromě přístupu k fotkám, souborům a médiím?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	NE
Prestigio PMT3777 3	v5.1	NE
Galaxy Nexus (*)	v7.1.1	NE
Pixel 2 (*)	v10.0	NE
Nexus S (*)	v11.0	NE

2. Neudělit oprávnění. Dostala Aplikace požadované oprávnění?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	NE
Prestigio PMT3777 3	v5.1	NE
Galaxy Nexus (*)	v7.1.1	NE
Pixel 2 (*)	v10.0	NE
Nexus S (*)	v11.0	NE

3. Spustit aplikaci. Vykreslila se obrazovka podle vzoru jednoho z obrázků I.2 a I.3?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

4. Zvolit tlačítko s ikonou fotoaparátu. Otevřel se fotoaparát?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

5. Vyfotografovat a počkat na výsledky. Zobrazily se výsledky?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

6. Zvolit tlačítko s ikonou rámečku. Otevřela se galerie obrázků?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

7. Zvolit obrázek. Načetl se vybraný obrázek a jsou vzápětí vidět i výsledky?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	NE
Prestigio PMT3777 3	v5.1	NE
Galaxy Nexus (*)	v7.1.1	NE
Pixel 2 (*)	v10.0	NE
Nexus S (*)	v11.0	NE

8. Zobrazil se dialog s chybovou hláškou?

Název zařízení	Android verze	Výsledek
Acer B1-760HD	v5.0.1	ANO
Prestigio PMT3777 3	v5.1	ANO
Galaxy Nexus (*)	v7.1.1	ANO
Pixel 2 (*)	v10.0	ANO
Nexus S (*)	v11.0	ANO

III Scénářové testování Webové aplikace

III.1 Testovací scénáře

III.1.1 První scénář

1. Povolit ve webovém prohlížeči JavaScript a neblokovat Cookies. Je to takto nastaveno?
2. Načíst url `http://192.168.1.132:8080/`. Vykreslila se webová stránka s formulářem pro nahrání obrázku?
3. Nahrát obrázek. Zobrazil se obrázek a vzápětí výsledky?

III.1.2 Druhý scénář

1. Zablokovat ve webovém prohlížeči JavaScript. Je to takto nastaveno?
2. Načíst url `http://192.168.1.132:8080/`. Vykreslila se webová stránka s chybovou hláškou (`If you want to see result, enable JavaScript please.`) a formulářem pro nahrání obrázku?
3. Nahrát obrázek. Zobrazil se obrázek?
4. Zmáčknout tlačítko `Show prediction`. Přibyla v zobrazení další chybová hláška? (`Cookies are blocked or JavaScript is disabled...`)

III.1.3 Třetí scénář

1. Zablokovat všechny Cookies. Je to takto nastaveno?
2. Načíst url `http://192.168.1.132:8080/`. Vykreslila se webová stránka s formulářem pro nahrání obrázku?
3. Nahrát obrázek. Zobrazila se místo obrázku a výsledků chybová hláška? (`Cookies are blocked or JavaScript is disabled...`)

III.2 Testovací report

První scénář

1. Povolit ve webovém prohlížeči JavaScript a neblokovat Cookies. Je to takto nastaveno?

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO
iOS	Firefox	v99.0	ANO
iOS	Google Chrome	v100.0.4896.85	ANO
iOS	Safari	v15.4.1	ANO
Android	Google Chrome	v43.0.2357.93	ANO

2. Načíst url `http://192.168.1.132:8080/`. Vykreslila se webová stránka s formulářem pro nahrání obrázku?

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO
iOS	Firefox	v99.0	ANO
iOS	Google Chrome	v100.0.4896.85	ANO
iOS	Safari	v15.4.1	ANO
Android	Google Chrome	v43.0.2357.93	ANO

3. Nahrát obrázek. Zobrazil se obrázek a vzápětí výsledky?

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO
iOS	Firefox	v99.0	ANO
iOS	Google Chrome	v100.0.4896.85	ANO
iOS	Safari	v15.4.1	ANO
Android	Google Chrome	v43.0.2357.93	ANO

Druhý scénář

---

1. Zablokovat ve webovém prohlížeči JavaScript. Je to takto nastaveno?

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO

2. Načíst url `http://192.168.1.132:8080/`. Vykreslila se webová stránka s chybovou hláškou (`If you want to see result, enable JavaScript please.`) a formulářem pro nahrání obrázku?

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO

3. Nahrát obrázek. Zobrazil se obrázek?

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	NE
Linux	Firefox	v99.0	NE
Windows 10	Firefox	v99.0.1	NE
Windows 10	Google Chrome	v100.0.4896.75	NE
Windows 11	Firefox	v99.0.1	NE
Windows 11	Opera	v85.0.4341.60	NE

4. Zmáčknout tlačítko `Show prediction`. Přibyla v zobrazení další chybová hláška? (`Cookies are blocked or JavaScript is disabled...`)

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO

Třetí scénář

1. Zablokovat všechny Cookies. Je to takto nastaveno?

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO

2. Načíst url `http://192.168.1.132:8080/`. Vykreslila se webová stránka s formulářem pro nahrání obrázku?

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO

3.Nahrát obrázek. Zobrazila se místo obrázku a výsledků chybová hláška? (`Cookies are blocked or JavaScript is disabled...`)

Operační systém	Webový prohlížeč	Verze	Výsledek
Linux	Opera	v85.0.4341.60	ANO
Linux	Firefox	v99.0	ANO
Windows 10	Firefox	v99.0.1	ANO
Windows 10	Google Chrome	v100.0.4896.75	ANO
Windows 11	Firefox	v99.0.1	ANO
Windows 11	Opera	v85.0.4341.60	ANO

IV Architektonické slohy

Zde rozebereme vlastnosti a použitelnost jednotlivých architektonických slohů pro jejich automatickou detekci.

IV.1 Starověký Egypt

Pro tento sloh jsou typickými stavbami například monumentální sochy nebo chrámy. Charakteristické prvky jsou na první pohled viditelné. Patří mezi ně masivní zdivo a těžké sloupy. Co se týče konkrétních staveb, těch už tolik nenajdeme. Nejznámějšími stavbami jsou pyramidy v Gíze, Abú Simbel (chrámový komplex na Jihu Egypta) [21] a Velká sfinga v Gíze. Většina z nich se nachází na území Egypta.

Obrázek IV.1: Vlevo: Abú Simbel (obrázek převzat z [18]), vpravo: Pantheon v Římě, příklad antické architekrury (obrázek převzat z [19])

IV.2 Antika (Řecko, Řím)

Známý sloh jehož charakteristické rysy se dělí na tři stylová období podle tvarů sloupů (dórský, ionský, korintský). Existuje poměrně rozvinutá nauka o architektuře, rozdělení a pojmenování jednotlivých prvků stavby. Mezi typy staveb patří chrámy, sochy, amfiteátry, hipodromy (stavby, kde se konaly koňské závody), urbanistické komplexy, vily, lázně či sochy. Konkrétní stavby jsou třeba akropolis, Pantheon, Koloseum anebo císařské vily na ostrově Capri.

IV.3 Románský sloh

Známé stavební prvky tohoto slohu jsou sloupy (zde je vidět návaznost na Antiku), masivní kamenné zdivo a oblouková klenba. Typické stavby jsou ku příkladu baziliky, rotundy, hrady nebo první městské struktury v Čechách (náměstí). Konkrétní známé baziliky jsou ve městě Cáchy, Špýr, Praha (bazilika sv. Jiří). Rotundy bychom mohli v Čechách hledat v Praze a jejím okolí nebo například ve Starém Plzenci u Plzně. Z hradů bychom mohli vyjmenovat třeba zříceninu Přimda, Chebský hrad, Landštejn a další.

Obrázek IV.2: Vlevo: Rotunda na hoře Říp (obrázek převzat z [37]), vpravo: katedrála Notre Dam v gotickém slohu (obrázek převzat z [2]))

IV.4 Gotika

Architekti gotického slohu jako Petr Parléř, Benedikt Rejt nebo Matěj Rejsek jsou známí stavbami chrámů, paláců, hradů, měšťanských domů či opevnění/hradeb. Stavební prvky charakterizující tento sloh jsou třeba velké klenuté prostory, lomený oblouk, opěrný systém a například trojlodní katedrální systém s věncem kaplí. Konkrétní stavby jsou známé po celém světě. Památky ku příkladu Notre-Dame de Paris, Westmister Abbey, chrám v Kolíně nad Rýnem, Katedrála sv. Víta na Pražském hradě, chrám sv. Bartoloměje v Plzni, sv. Petr a Pavel v Brně, sv. Barbora v Kutné Hoře, Vladislavský sál na Pražském hradě, hrady Pernštejn, Karlštejn, Švihov nebo Dům U Zvonu na Staroměstském náměstí.

IV.5 Renesance

Renesanční sloh má mnoho představitelů. Výčet těch nejznámějších obsahuje jména jako třeba Michelangelo Buonarotti, Leonardo da Vinci, Bonifác Wohlmuth, Paolo de La Stella či Baldassare Maggi. Kromě typů staveb z minulé sekce IV.4 se zde objevují i zámky a letohrádky. Typickou stavbou ze

zahraničí je Bazilika sv. Petra v Římě. V českých zemích pak Letohrádek královny Anny (Belveder), radnice v Plzni a Kolíně, zámky v Telči, Litomyšli, Pardubicích, Březnici, zámek Kratochvíle a Opočno, domy na náměstí ve Slavonicích. Většina staveb není církevního charakteru. V českém prostředí jsou pro renesanci známá psaníčková sgrafita na fasádách zámků a domů.

Obrázek IV.3: Vlevo: Plzeňská radnice v renesančním slohu (obrázek převzat z [34]), vpravo: barokní chrám Karlskirche ve Vídni

IV.6 Baroko

Pro tento sloh je typická dynamická a ohromující architektura, důraz na iluzionismus, používání soch na stavbách, tvorba krajinného prostředí, důraz na duchovní stránku architektury. Česká a moravská města a do značné míry i kulturní krajina mají převážně barokní charakter na gotických a renesančních základech. Pro úplnost zmíníme několik představitelů baroka, jako je například Kryštof Dientzenhofer, Kilian Ignách Dientzenhofer, Carlo Lurago, Giovani Batista Alliprandi. Památky najdeme nejen v Praze, kde se nachází Chrám sv. Mikuláše či Křižovnický kostel, ale i po celé České Republice a Evropě.

IV.7 Secese

Osobitý sloh s důrazem na provedení řemesla. Od předchozích slohů se výrazně odlišuje ornamentálností a rovnými liniemi na stavbách. Z Českých památek můžeme jmenovat Obecní dům v Praze, Národní dům Prostějov nebo Hlavní nádraží v Praze. Známými architekty Českých zemí jsou například Jan Kotěra, Josef Balšánek a Josef Fanta.

Obrázek IV.4: Vlevo: Secesní budova Hlavního nádraží (převzato z [40]), vpravo: funkcionalistický Veletržní palác v Praze (obrázek převzat z [46])

IV.8 Funkcionalismus

Modernistický sloh s důrazem na praktičnost stavby, což jsou veřejné budovy, městské bytové domy, vily nebo administrativní budovy. Má funkční a přehledné prvky, například přímé pravoúhlé fasády. České stavby ve slohu funkcionalismu jsou třeba Veletržní palác v Praze, Penzijní ústav na Žižkově, škola Vesna v Brně, vila Tugendhat v Brně a Mullerova vila v Praze. Mezi představitele patří Le Corbusier, Mies van Den Rohe, Walter Gropius, Bohuslav Fuchs, Oldřich Tyl a Josef Gočár.

IV.9 Kubismus

Pokud bychom přemýšleli o typicky českém stavebním slohu, mohl by to být právě kubismus, který se pyšní prvky jako je výrazná plasticita, více pohledové geometrické ztvárnění, lomené tvary, které se inspirují v malířství a sochařství. Dům u Černé matky Boží v Celetné ulici v Praze nebo vily pod Vyšehradem se řadí do památek tohoto slohu. Významní architekti jsou Josef Gočár, Josef Chochol nebo Pavel Janák.

IV.10 Brutalismus

Název tohoto slohu je odvozen ze slova brut neboli drsný beton. Typické budovy jsou například bytové domy a veřejné budovy. Konkrétní příkladem je kaple ve Francii, obytný dům v Marseille, Hotel Thermal v Karlových Varech nebo pražské budovy Transgas a Centrotex.

Obrázek IV.5: Vlevo: Dům u Černé matky Boží ve slohu kubismu (převzato z [10]), vpravo: Příklad brutalismu (obrázek převzat z [4])

IV.11 Islámská a arabská architektura

Hojně rozšířený sloh, ať už se jedná o všeobecné povědomí mezi lidmi, nebo skutečnost, že architektonické památky a budovy (hlavně ty náboženské) najdeme skoro po celém světě.

Za charakteristické prvky můžeme považovat klenby a kupole. Druhy staveb jsou například paláce či náboženské budovy. Konkrétním příkladem paláce by mohla být například Alhambra ve Španělsku či pevnost Ágra v Indii. Co se náboženských budov týče, můžeme jmenovat Tádž Mahal, který se také nachází ve městě Ágra, nebo mešita sultána Ahmeda v Istanbulu.

Obrázek IV.6: Alhambra ve španělském městě Granada (převzato z [41])

IV.12 Indická a khmérská architektura

Velice starobylý a v Evropě poměrně neznámý sloh. Typické budovy lze najít na území dnešní Kambodži. Mezi stavební materiály patřily cihly, pískovec nebo laterit, což je druh kamene. Hlavním poznávacím prvkem je takzvaný prang. Je to vysoká a většinou bohatě zdobená věž.

Obrázek IV.7: Vlevo: Příklad Khmérské stavby (obrázek převzat z [11]), vpravo: Příklad chrámu v Japonsku (obrázek převzat z [44])

IV.13 Slohy staré Číny, Japonska a Koreje

Navzájem velmi podobné slohy se společnými typickými prvky jako například důraz na symetrii, zakřivené krokve (nosné konstrukce střechy) typické nejen pro takzvané pagody (náboženské budovy). Hlavním materiálem je dřevo a případně cihly. Mezi typické stavby patří paláce, obytné domy nebo chrámy.