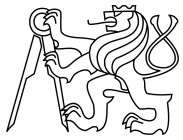
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Rozpoznávání památek
Student:	Bc. Peter Bábics
Vedoucí:	Ing. Pavel Kordík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Nastudujte metody rozpoznávaní obrazu pomocí neuronových sítí. Připravte soubor obrázků pražských památek získaný z internetu nebo jejich nafocením. Připravte trénovací data do podoby zpracovatelné a vhodnou metodou zkonstruujte model pro rozpoznávání památek například pomocí předtrénované hluboké neuronové sítě, kde přiučíte výstupní vrstvu rozpoznat jednotlivé památky (tedy přiřadit fotografii ke správné památce). Zhodnoťte kvalitu rozpoznávání na obrázcích z mobilního telefonu pomocí křížové validace.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 4. dubna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Teoretické Informatiky

Diplomová práca

Rozpoznávání památek

Bc. Peter Bábics

Vedúci práce: Ing. Pavel Kordík, Ph.D.

9. mája 2017

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 9. mája 2017 .....

Odkaz na túto prácu

Bábics, Peter. *Rozpoznávání památek*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

V tejto práci som sa zameral na analýzu existujúcich prístupov pri rozpoznávaní objektov z fotografií. Analyzoval som možné zdroje dát pre historické budovy v Prahe, vhodným spôsobom som vytvoril súbor fotografií pre strojové učenie, následne som testoval rôzne modely a prístupy pri rozpoznávaní pamiatok, využívajúce hlboké konvolučné siete. Nakoniec som vybraný model implemetnoval do jednoduchého frameworku, ktorý som následne napojil na chatbota *Golem* a otestoval celkovú integráciu pomocou mobilnej aplikácie *Messanger*

Klíčová slova Strojové učenie, Strojové videnie, Rozpoznávanie, Klasifikácia, Neurónové siete, Konvolúcia, Hlboké konvolučné siete, Chatbot, Golem

Abstract

I analyzed existing approaches to recognizing objects from photos. I analyzed the possible sources of data for historical buildings in Prague, and in a suitable way I created a set of photographs for machine learning, then I tested various models and approaches for recognizing historical sites using deep convolutional networks. Finally, the selected model was implemented into a

simple framework, which I then connected to the chatbot *Golem* and tested overall integration using the mobile application *Messanger*

Keywords Machine Learnning, Computer vision, Recognition, Classification, Neural network, Convolution, Deep convolutional networks, Chatbot, Golem

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Úlohou tejto práce je oboznámiť sa s problematikou klasifikácie objektov z obrázkov pomocou neurónových sietí. Vhodne zvoliť, prípadne vytvoriť neurónovú sieť schopnú klasifikovať pamiatky z obrázkov poskytnutých užívateľom a napojenie tohoto systému do chatbota *Golem*, čím sa umožní vstup zo smartfónovej aplikácie *Messanger*. Získané dáta potrebné na trénovanie a obecne validáciu mnou použitého modelu, musia byť robustné. Musia obsahovať informácie o pamiatke z rôznych uhlov a pozíc, pri rôznych svetelných podmienkach, natočení kamery, rozostrenia obrazu, preto aby bolo zaručená klasifikácia nech už stojí užívateľ kdekoľvek, fotí z akéhokoľvek uhlu a v akýchkoľvek reálnych podmienkach.

Kapitola 1

Cieľ práce

Rozpoznávanie objektov z fotografií je jednou z najťažších úloh strojového učenia, ktorá má za cieľ nájsť na obrázku jednotlivé objekty a kategorizovať. Čiže sa jedná o viacnásobnú klasifikáciu / viac triednu klasifikáciu. Cieľom mojej práce je vytvoriť súbor fotografií pražských pamiatok získaním z internetu alebo ich nafotením, následne ich predspracovať, vytvoriť model pre ich rozpoznávanie založený na predtrénovanej hlbokej neurónovej sieti a tento model overiť pomocou krížovej validácie. Naviac pre reálne sprístupnenie som sa rozhodol tento model sprístupniť pomocou jednoduchého API a napojiť ho pomocou Chatbot Frameworku *GOLEM* na *Facebook chat* (*Messanger*).

Kapitola 2

Predchádzajúce prístupy

Problém úlohy rozpoznávania je známy niekoľko rokov. Za tento čas bolo vymyslených veľa prístupov a algoritmov pre riešenie tohoto problému. V posledných rokoch však pri riešení tohoto problému začínajú dominovať neurónové siete, konkrétne hlboké konvolučné siete. Obecne sa algoritmy pre riešenie tohoto problému dajú kategorizovať do dvoch skupín. Prvou skupinou sú algoritmy, ktoré porovnávajú celý obrázok, čiže generujú príznaky pre obrázok ako jeden celok. Druhou skupinou sú algoritmy, ktoré generujú príznaky pre časti obrázku.

Prvý typ algoritmov (generujúcich príznaky pre obrázok ako celok) môžeme rozdeliť do pár štandardných krokov strojového učenia. Tými sú, predspracovanie, extrakcia príznakov, natrénovanie klasifikátoru a validácia. Ako klasifikátory sa tu najčastejšie používajú *K-NN*, *SVM* [1] a *hlboké konvolučné siete*.

Druhý typ algoritmov využíva tie isté kroky, no predspracovanie je v tomto prípade nájdenie takzvaných *Interesting points*, čo sú body prípadne regióny v obrázkoch, ktoré ho najviac vystihujú. Tieto regióny sú zadané pozíciou v obrázku x,y ose, veľkosťou σ a orientáciou δ. Extrakcia príznakov sa následne počíta pre každý región zvlášť. Najčastejšie sa môžeme stretnúť so *SIFT* [2], *SURF* [3], *HOG* [4], *GIST*. Týmto dostávame súbor vektorov príznakov popisujúcich jednotlivé regióny obrázku a tým pádom aj samotný obrázok. Zostáva nám teda len porovnať získané príznaky s uloženou databázou, čo sa najčastejšie rieši výpočtom pomocou euklidovskej vzdialenosti.

2.1 Hľadanie *Interesting points*

Ako som už spomenul, prvou fázou rozpoznávanie objektov je nájdenie *Intersting points* na obrázku. Tieto body sú dané pozíciou v obrázku, škálou regiónu, ktorý popisujú a natočením. Na nájdenie a popísanie týchto bodov slúžia detektory (takzvané *Blob detector*). Táto časť algoritmov je riešená v

každom algoritme odlišne. Napríklad, *SIFT* [5] využíva *Difference of Gaussians (DoG)*, *SURF* [3] zas využíva detektor založený na *Hessian Matrix*.

2.1.1 SIFT - Difference of Gaussians (DoG)

Ako už z názvu vyplýva, tento detektor využíva rozdiel gausiánov obrázku. Obrázok najprv rozdelíme na oktávy. Každá oktáva zdvojnásobuje škálu σ a zmenšuje obrázok. Podľa experimentov v [2], vychádza ako najlepšie rozdelenie jednej oktávy na tri škály, pre ktoré sa následne počíta rozdiel v rastúcom poradí. Na každú škálu je následne aplikovaná konvolúcia s gausovským rozdelením, čo nám vygeneruje 3 obrázky, ktoré následne môžeme od seba odčítať (po pixloch). Následne pre každý rozdiel nájdeme minimum a maximum, tieto body sú pre nás zaujímave.

Po úvodnej detekcií nasleduje interpolácia získaných bodov na ich správne miesto v pôvodnom obrázku (túto časť robíme kvôli zmenšeniu obrázku pri zmene oktávy), toto je riešené využitím Taylorovho rozvoja. Týmto získame body so správnymi pozíciami.

Nakoľko ale týchto bodov je veľa, je potrebné ich ešte ďalej filtrovať. Dosiahneme toho odstránením bodov s nízkym kontrastom. Kontrast počítame cez druhý stupeň Taylorovho rozvoja rozdielu gausiánov. Ak je hodnota kontrastu menšia než 0.03 daný bod zahadzujeme.

Následne ešte odstránime body, ktoré boli detekované popri hranách pomocou vlastných čísel hesiánu.

A vo finálnej fázy určíme orientáciu. Táto je určená pomocou tangenty rozdielov gausiánov.

2.1.2 SURF - Hessian Matrix

Algoritmus SURF ako detektor pre *Interesting Points* využíva Hesián. Algoritmus opäť pracuje cez rôzne škály, no v tomto prípade pôvodný obrázok zostáva nezmenený a pracuje sa akurát s jeho výsekmi. Pre každý bod je najprv spočítaný jeho Hesián, podľa vzťahu

$$H(x, y, \sigma) = \begin{pmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{yx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \end{pmatrix}$$

Kde L_{xx} je druhá derivácia v xx smere, čiernobieleho obrázku, na ktorý bol aplikovaný gausovský filter s parametrom σ.

Z tohoto hesiánu vypočítame determinant a vyberáme body, pre ktorý je tento determinant maximálny. Následne je ešte potrebné priradiť orientáciu. Riešené je to pomocou natáčania detekovaného regiónu v $\frac{\pi}{3}$ uhle a hľadania maximálnej hodnoty deskriptoru pre daný bod (v závislosti na natočení).

2.2 Generátor príznakov - Deskriptor

Ďalšou dôležitou súčasťou rozpoznávania objektov z obrázkov je generátor príznakov, takzvaný deskriptor. Deskriptor generuje vektor príznakov (zvyčajne väčší nez 10 prvkov), ktorý sa snaží čo najlepšie popísať obrázok, vzhľadom na jas, textúry a iné. Deskriptory často bývajú súčasťou algoritmov, no dajú sa vytrhnúť z kontextu a použiť na obrázok ako celok, pri použití správnych parametrov. Príkladom takého deskriptoru je napríklad SURF deskriptor.

2.2.1 SIFT

Deskriptor SIFT-u je počítaný vždy pre všetky nájdené regióny separátne. Každý región je určený pozíciou, škálou a orientáciou. Tieto regióny sa rozdelia na 4x4 podregióny. Následne sa pre každý podregión spočíta histogram orientácií po ôsmych košoch (bins). Výsledný vektor sa ešte normalizuje na jednotkový, čo odstraňuje afínne transformácie osvetlenia a uplatní sa threshold funkcia s hodnotou 0.2 na odstránenie nelineárneho osvetlenia. Čo nám vo výsledku dáva 128 položkový vektor popisujúci *Interesting point*.

2.2.2 SURF

SURF deskriptor funguje podobným štýlom ako SIFT, avšak v tomto prípade sa región okolo *Interesting point*, rozdelí na 4x4 podregióny, ktoré sa následne rozdelia na 5x5 rovnomerne rozdelených rámčekov. Každý podregión následne vypočíta Haarove wavelety [6] v X a Y smere pre každý svoj rámček. Hodnoty X a Y sa sčítajú separátne, taktiež vytvoríme súčet kvadrátov týchto hodnôt. Toto je vykonané pre všetky podregióny. Následne všetky hodnoty uložíme do vektora, ktorý normalizujeme na jednotkový. Dostaneme 64 prvkový vektor príznakov. Táto metóda však môže generovať aj dlhšie prípadne kratšie vektory príznakov. Dĺžka primárne záleží od voľby rozdelenia na podregióny, napríklad SURF-36 využíva 3x3 podregióny. Taktiež je možne využiť ďalšie hodnoty pri počítaní príznakov jedného podregiónu, ako napríklad súčet absolútnych hodnôt, stredných hodnôt, priemerov a iné.

2.2.3 Histogram of Oriented Gradients - HOG

Prvým krokom vo výpočte HOG je výpočet gradientu v X a Y ose. Následne vypočítame smer a veľkosť gradientu v každom bode. Ďalším krokom je výpočet histogramu gradientov v 8x8 mriežke. Pre každý prvok mriežky vytvoríme histogram gradientu s 9 košmi. A na záver je obrázok ešte raz rozdelený. Tentokrát 16x16 mriežke a jednotlivé histogramy, ktoré spadajú do tejto mriežky sú normalizované, čo nám zaručuje istú invarianciu voči osvetleniu.

2.3 Vyhľadávanie

V predchádzajúcich kapitolách som opísal hľadanie a generovanie príznakov pre výseky obrázkov. V tejto kapitole popíšem najčastejšie prístupy v efektívnom vyhľadávaní zhôd.

Pokiaľ máme vysoké množstvo obrázkov (rádovo v miliónoch), počet príznakov na jeden obrázok môže byt niekoľko stoviek, tisícov. Pre efektívne vyhľadávanie potrebujeme počet týchto hodnôt zmenšiť. Počet príznakov môžeme zmenšovať pomocou štandardných metód *Feature Selection / Feature Extraction*. Táto kapitola sa však venuje použitiu klasteringu pre urýchlenie výpočtu podobností. V článkoch som sa stretol s dvoma podobnými prístupmi. Prvým z nich je štandardné K-Means, druhým bolo Hierarchic K-Means. Obecne je možné použiť akúkoľvek metódu clusteringu, ktorá udáva "všeobecnú reprezentáciu"svojich clusterov.

2.3.1 K-Means

Štandardným prístupom v minimalizácií počtu záznamov je K-Means. Jedná sa o iteratívny algoritmus. Na počiatku sú náhodne určené stredy (k - stredov). Následne sa každému stredu priradia jeho najbližší susedia (štýlom 1-NN). Pomocou nich sa následne vypočíta nová pozícia stredu a ten sa posunie na ňu. Toto prebieha pre zvolený počet iterácií, alebo pokiaľ nenastane stabilný stav (pohyb stredov je nulový, prípadne minimálny).

Využitie pri rozpoznávaní objektov je nasledovné. Najprv vypočítame štandardný K-means algoritmus. Následne každému stredu, priradíme "značky"všetkých záznamov, ktoré pod tento stred spadajú a prípadne ich vzdialenosti, ktoré môžu byť použité na váženú klasifikáciu. Vyhľadávanie záznamu spočíva vo výpočte vzdialeností príznakov od jednotlivých stredov. Následne sa vyberú najbližšie stredy, z nich zistíme značky, z ktorých spravíme sumu, prípadne váženú sumu. Nakoniec vyberieme obrázok ktorý má najviac podobných značiek.

Možným rozšírením tohoto algoritmu je využitie *thresholdovej funkcie*, ktorá nám umožní nepriradiť testovaný prvok žiadnemu z netrénovaných a tak umožniť určenie neznámych objektov. Využité napríklad bolo v práci [7].

2.3.2 Hierarchic K-means

Hierarchické K-Means je podobné štandardnému K-Means. Rozdiel je však v tom, že využívame nižšiu hodnotu parametra K, Algoritmus je veľmi podobný štandardnému K-means, ako prvé urobíme štandardné K-means na celej množine, následne rekurzívne opakujeme pre všetky získané klastre, do požadovanej hĺbky, prípadne počtu značiek na list.

Vyhľadávanie je taktiež skoro rovnaké, avšak v tomto prípade rekurzívne zostupujeme dokiaľ neprídeme do listu. Využité napríklad bolo v práci [8].

2.4 Rozpoznávanie pomocou Neurónových sietí

V posledných rokoch sa do hry zapojili neurónové siete, špecificky hlboké neurónové siete a konvolučné neurónové siete. Od roku 2 010 každoročne prebieha súťaž *ImageNet Large Scale Visual Recognition Challenge*, v ktorej súťažia jednotlivé algoritmy v rozpoznávaní objektov na obrázkoch. Posledné roky dominujú hlboké neurónové siete. Ako príklady uvediem R-CNN [9], AlexNet [10], VGGNet [5], ResNet [11] a DenseNet [12]. Tieto siete boli trénované na dátach, na ktorých boli vyznačené jednak zaujímavé regióny (*Region of Interest - RoI*) a taktiež boli tieto regióny popísané (*labeled*).

Tieto neurónové siete sa skladajú z konvolučných vrstiev, nad ktorými je zväčša postavený viacvrstvový perceptron. Takýto typ sietí je možné učiť štandardným *SGD* s využitím *Back propagation* algoritmu.

2.4.1 AlexNet

Sieť *AlexNet* je výhercom *ILSVRC* z roku 2 012, kedy vyhrala v rozpoznávaní objektov s top-5 chybou 16 %, druhá sieť mala chybu 26 %. Top-5 chyba je určená ako správna klasifikácia v najpravdepodobnejších 5 výsledkoch. Ako konvolučná sieť je tu použitá sieť skladajúca sa z piatich konvolučných vrstiev. Prvá vrstva má na vstupe obrázok o rozmeroch 224x224x3. Spracuje ho s 96 kernelmi o veľkosti 11x11x3 so striedou 4 pixely. Výstup vrstvy sa normalizuje a je naňho aplikovaný max pooling. Výstup je následne vedený do druhej konvolučnej vrstvy s 256 kernelmi o rozmeroch 5x5x48. Nasledujúce tri konvolučné vrstvy sú prepojené bez úpravy výstupu (t.j bez normalizácie, poolingu a ReLU). Tretia vrstva má 384 kernelov o rozmeroch 3x3x256, štvrtá taktiež 384 kernelov, tentokrát o rozmeroch 3x3x192 a koncová piata vrstva má 256 kernelov o rozmeroch 3x3x192. Následne je na výstup aplikovaný max pooling.

Výstup max poolingu je vedený do MLP siete s 4 096 neurónmi v skrytej vrstve. Táto sa stará už o klasifikáciu, na učenie bol využitý štandardný *SGD*, spolu s *backpropagation*.

2.4.2 R-CNN

R-CNN sa skladá z dvoch častí, detektoru zaujímavých regiónov (*RoI*) a klasifikátora. Zdieľa konvolučnú sieť medzi detektorom a klasifikátorom. Detektor zaujímavých regiónov (taktiež nazývaný *Region Proposal Network - RPN*). Ako konvolučná sieť je tu použitá sieť z AlexNet. Za konvolučnou sieťou sa nachádza pohyblivé okienko, ktoré je vstupom do neurónovej siete určujúcej, či sa v ňom nachádza objekt. Výstup tejto neurónovej siete, spolu s s pozíciou okienka je uložený pre neskoršie spracovanie. Akonáhle je vypočítaná pravdepodobnosť výskytu objektu vo všetkých okienkach, sú tieto okienka zlúčené tak aby sa maximalizovala veľkosť okienka pri zachovaní minimálneho pokry-

tia zvoleného objektu. Týmto sú vygenerované jednotlivé regióny objektov. Následne vyextrahujeme z konvolučnej siete mapu príznakov (*feature map* - je výstupom konvolučnej siete) vo zvolených regiónoch, vhodne ju naškálujeme a dáme ju na vstup klasifikačnej neurónovej siete.

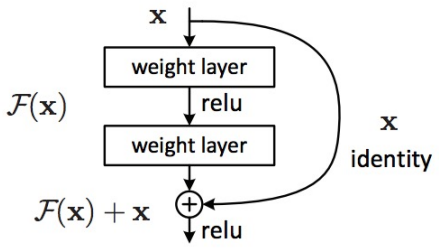
2.4.3 VGGNet

Jedná sa o neurónovú sieť, ktorá zvíťazila v *ILSVRC* v roku 2 014. Využíva 16 konvolučných vrstiev zakončených plne prepojenými vrstvami, pričom konvolučné vrstvy sa dajú rozdeliť na určité štádiá. Všetky využívajú rovnaký rozmer kernelu, 3x3. Mení sa len počet kernelov, pričom v každej fáze sa zdvojnásobuje. Začíname na 64, po dvoch konvolučných vrstvách s 64 kernelmi a rozmermi 3x3. Následuje max pool, ďalšie štádium obsahuje ďalšie dve konvolučné vrstvy, tento krát s 128 kernelmi, za nimi sa opäť nachádza max pool vrstva. Nasledujú štyri konvolučné vrstvy s 256 kernelmi, max pool vrstva. Posledné dve konvolučné vrstvy majú veľkosť kernelu 512 a nachádzajú sa za nimi max pool vrstvy. Výstup poslednej max pool vrstvy je vedený do MLP s 4 096 neurónmi v skrytej vrstve.

Zaujímavosťou tejto neurónovej siete je, že využíva kaskádu konvolučných vrstiev, vždy s rovnakými rozmerom, pričom vždy v štádiu je rovnaký počet kernelov. Nevýhodou takéhoto riešenie je však výpočetná náročnosť.

2.4.4 ResNet

Ďalšou neurónovou sieťou ktorú spomeniem je ResNet[11]. Táto neurónová sieť zvíťazila v *ILSVRC* v roku 2 015. Jej hlavnou ideou je využitie takzvaných *shortcut links*, čo je v podstate vrstva, ktorá k aktuálnemu výsledku pripočíta hodnotu z predchádzajúcej vrstvy, čiže inak vyjadrené: $g(\hat{x}) = f(\hat{x}) + \hat{x}$, kde $g(x)$ je výsledok aktuálnej vrstvy a $f(x)$ je výstup predchádzajúcej. Pre lepšie znázornenie prikladám obrázok 2.1.



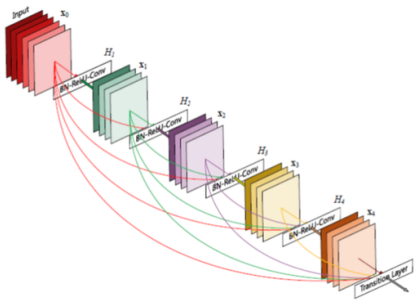
Obr. 2.1: *Shortcut link* ako blok neurónovej siete ResNet

Je možné pripočítať aj hodnotu z n-tej predchádzajúcej vrstvy, ako bolo použité v samotnej sieti použitej v súťaži *ILSVRC*, kde táto sieť dosiahla top-5 chybu rovnú 3.57 %. Jej návrh je podobný ako návrh VGGNet [5].

Využíva rovnaké princípy v zmysle hĺbky, rozdelenia na fázy, v ktorých sú rovnaké konvolučné vrstvy spojené za sebou. Zmena ktorú prináša je viac vrstiev vrámci jednej fázy a *shortcut links* vždy vedúce medzi každou druhou vrstvou, pričom za každým súčtom je *ReLU* aktivačná funkcia. Nakoľko medzi prechodmi medzi jednotlivými fázami dochádza k redukcií dimenzionality, je potrebné upraviť výstupy tak aby bol súčet s predchádzajúcimi hodnotami možný. Článok uvádza niekoľko metód, ako doplnenie výstupov nulami, "pripojenie"predchádzajúcich hodnôt na výstup (pridanie na koniec vektoru hodnôt), transformáciu predchádzajúcich hodnôt (lineárna / nelineárna) ako doplnenie nulami a následne pričítanie.

2.4.5 DenseNet

Poslednou neurónovou sieťou ktorú spomeniem je DenseNet [12]. Postavená je na základe *ResNet-u*, ktorý rozširuje o takzvané *DenseBlock*-y, ktoré obsahujú viac *Shortcut Links*, s tým, že súčty sa robia s každým podblokom, najlepšie toto vyjadruje obrázok 2.2.



Obr. 2.2: *DenseNet blok* ako blok neurónovej siete DenseNet

Sieť *DenseNet* je zaujímavá ako rozšírenie ResNet-u, síce jej top-5 chyba je horšia (6.21 %), no stále je dosť vysoká a pre účely tejto práce by mala postačovať.

Kapitola 3

Analýza a návrh

Kapitola ako už z názvu vyplýva sa zaoberá analýzou problému rozpoznávania historických pamiatok. Postupne si popíšeme jednotlivé úskalia od hľadania zdrojov fotografií, až po výber modelu a krížovú validáciu a samotnú implementáciu frameworku a integráciu detektoru pamiatok doňho.

3.1 Hľadanie zdrojov fotografií

Už samotný fakt, že hľadáme fotografie vyvoláva otázky odkiaľ ich získame. Vzhľadom na typ úlohy by sme chceli súbor obrázkov čo najväčší, aby pokryl danú budovu z rôznych strán, uhlov a pozíc. Existujú síce súbory fotografií pamiatok, no tieto sú z miest ako Madrid, Paríž a ďalšie. Takýto súbor pre Prahu neexistuje. Bol nútený vytvoriť si vlastný súbor obrázkov, získaných, či už z internetu alebo nafotením. Väčšina historických budov síce má svoju webovú stránku, na ktorej majú krátky zoznam fotografií. Fotografie poväčšinou obsahujú zopár záberov exteriéru ale prevážna väčšina z nich sú fotografie interiéru. Takže tieto fotky sú pre nás použiteľné ale nedostatočne obsiahle.

3.1.1 Google Street View

Ďalším zdrojom fotografií je *Google Street View*. Tento zachycuje dané objekty z rôznych pozícií a vďaka tomu, že fotografie, ktoré obsahuje sú panoramatické, môžeme programovo vytvoriť súbory jednoduchým zvolením iného žiadaného uhlu. Nevýhodou *Google Street View* je však limit na počet dotazov na server, ktorý činí 25 000 dotazov na deň na jeden API kľúč, ďalšou nevýhodou je skutočnosť, že na získanie fotografie musí užívateľ zadať koordináty v zemepisnej šírke a dĺžke, pričom nemá zaručené, že sa panoráma na daných koordinátoch nachádza. Ďalšou nutnosťou manuálneho spracovania výsledného získaného súboru, nakoľko dané fotografie môžu byť poškodené alebo prípadne neobsahujú zvolenú pamiatku, nakoľko medzi pozíciou fotenia a cieľovou pamiatkou sa nachádza budova. Ako poslednú a hlavnú nevýhodu spomeniem fakt, že

daná historická pamiatka musí byť blízko pri ceste, keďže fotografie sú vytvárané priechodom mapovacieho auta alebo bicyklu.

Pri *Google Street View* je teda potrebné zvoliť pamiatky tak, aby boli blízko cesty. Zakódovať ich pozíciu ako zemepisnú šírku a dĺžku. Následne nájsť pozície na ktorých sa dané fotografie nachádzajú, stiahnuť ich a manuálne spracovať.

3.1.2 Google Custom Search (Image search)

Ďalším zdrojom je *Google Custom Search*, ktorý poznáme ako *Google Image Search*. Ten umožňuje zadať textový popis hľadaného objektu, s tým, že vráti zoznam relevantných odkazov, v našom prípade fotografií. Tieto fotografie sú popísané slovne, pričom popis je získaný z kontextu zdroja. Môže sa stať, že daná fotografia bude zle popísaná, prípadne, že patrí k inému kontextu v prípade, že sa mieša viac kontextov na stránke. Ďalej toto API vracia 10 záznamov na stránku, pričom denný limit je 100 dotazov na API kľúč. Ďalšou nevýhodou je skutočnosť, že vrátený zoznam nie je statický a nachádzajú sa v ňom duplicity.

Podobne ako pri *Google Street View* aj pri *Google Custom Search* je potrebné vhodne zvoliť popis hľadaných objektov, vytvoriť zoznam, stiahnuť ich a manuálne spracovať kvôli výberu vhodných obrázkov.

3.1.3 Vlastné fotografie

Posledným zvoleným zdrojom sú fotografie, vytvorené pri vybraných pamiatkach. Fotografie by mali byť vyfotografované z miest, na ktorých turisti bežne fotografujú, prípadne zastavujú. Taktiež rôzne zaujímave pozície a pohľady na pamiatku. Fotografie by mali obsahovať daný objekt v rôznych pozíciach výslednej fotografie, pod rôznymi uhlami, pod rôznym natočením fotoaparátu a za rôznych svetelných podmienok. Cielom je vytvoriť súbor dát obsahujúci rôzne scenáre pod ktorými môže byť fotografia vyfotografovaná. Nakoľko pre účely strojového učenia z fotografií je potrebné veľké množstvo fotografií, výhodnejším spôsobom je natočenie videa danej pamiatky a následná extrakcia jednotlivých snímkov. Počas natáčania by mal objektív postupne zachytávať rôzne časti zvoleného objektu, pričom pohyb by mal byť pomalý aby vzniknuté snímky neboli rozmazané. Mal by postupne natáčať pamiatku z hora dolu a z ľava do prava, tak aby každá časť pamiatky na videu postupne bola zachytená vo všetkých okrajoch fotografie. Taktiež počas natáčania je možne jemne natáča kameru aby vznikli ďalšie unikátne zábery.

Vytvorené video zvoleného objektu pri štandardnom snímkovaní kamier (24 snímkov za sekundu) by po prvej minúte malo obsahovať 1440 snímkov. Keďže potrebujeme taktiež pamiatku mať na fotografiách z rôznych uhlov, rôzne jej časti a s rôznymi svetelnými podmienkami, je potreba týchto videí natočiť niekoľko.

3.2 Výber pamiatok

Dôležitým krokom je výber vhodných pamiatok. Požiadavky na výber pamiatok sú závislé primárne na použitých zdrojoch, nakoľko napríklad *Google Street View* vyžaduje aby bola daná historická budova dobre dostupná pre autá, prípadne bicykel. *Google Custom Search* zas vyžaduje aby bola pamiatka známa, čím známejšia, tým viac ľudí zverejňuje jej fotografie. Z tohoto dôvodu som vylúčil menej známe pamiatky, taktiež pamiatky, pri ktorých nie je priamo cesta ako sú napríklad sochy. Pamiatky, ktoré sa nedajú zachytiť jednou fotografiou ako je *Zlatá Ulička*. Prípadne majú veľa súčastí ako napríklad *Karlův most*.

Zoznam vybraných pamiatok 3.1 je písaný v angličtine z dôvodu jednoduchšieho vyhľadávania a popisu. Obe *Google* služby si daný názov prevedú do svojej internej reprezentácie.

3.3 Filtrovanie fotografií

V predchádzajúcich častiach bolo spomenuté, že všetky fotografie stiahnuté z internetu vyžadujú filtráciu, nakoľko sa môže stať, že neobsahujú presne vybranú pamiatku, obsahujú interiér, obsahujú viac pamiatok, alebo neobsahujú žiadnu.

Samotné filtrovanie je potreba vykonať ručne, lebo dopisiať neexistuje žiadny klasifikátor na pražské pamiatky. Môj návrh na filtrovanie fotografií z *Google Street View* využije jednoduchú aplikáciu, ktorá by postupne zobrazovala fotografie, zobrazovala informácie z nich ako sú vzdialenosť od objektu, uhol pod ktorým je fotografia vyfotografovaná, popis čo ma zvolená fotografia obsahovať a nakoniec obsahovala tlačítka pre určenie, či daný slovný popis sa zhoduje s objektom na fotografií. Takáto aplikácia by mala urýchliť spracovanie v prípade že počet fotografií bude vyšší než pár desiatok, nakoľko mnou používaný súborový manažér nie je vhodný pre zobrazovanie stoviek a viac súborov v jednom adresári.

Na filtrovanie fotiek z *Google Custom Search* by mal postačiť súborový manažér, vzhľadom na limit dotazov, ktorý táto služba má a skutočnosť, že vrátené odkazy môžu byť vysoko duplicitné.

3.4 Návrh modelu klasifikátoru

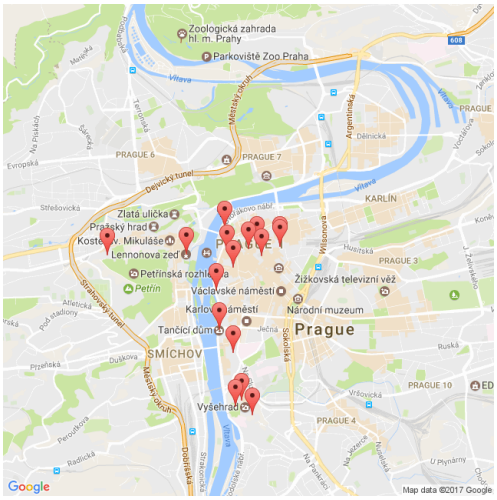
Samotný návrh modelu je jednoduchý. Na začiatku máme obrázok, tento najprv predpripravíme zmenšením na veľkosť potrebnú pre ďalšie spracovanie. Následne sa obrázok predá predtrénovanej neurónovej sieti pre generovanie príznakov. Získaný vektor príznakov môžeme ďalej zmenšiť pomocou výberu príznakov, prípadne analýzy hlavných komponentov. Výsledný vektor príznakov ďalej predáme vybranému modelu pre klasifikáciu, pokiaľ výstup modelu

- Lennons Wall
- Emauss
- Clementinum
- The Powder Tower
- National Theatre
- Dancing House
- Strahov Monastery
- Rudolfinum
- The Bethlehem Chapel
- The Estates Theatre
- Leopold Gate
- Brick Gate
- Basilica of St. Peter and St. Paul, Prague
- Prague astronomical clock
- Church of Our Lady before Tyn
- The Municipal House

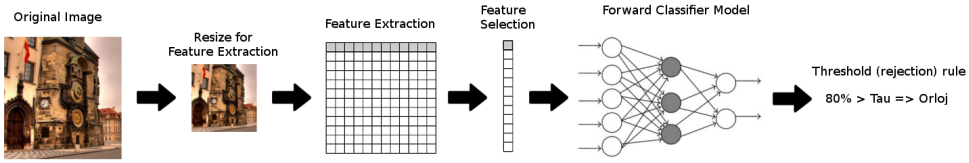
Obr. 3.1: Zoznam vybraných pamiatok

nie je jednoznačný, napríklad pravdepodobnosť priradenia do danej triedy je menšia než vopred zvolené τ nepriradíme daný obrázok žiadnej triede a nazveme ho neznámym. Celý návrh modelu je možné lepšie vyčítať z obrázku 3.3.

Návrh umožňuje modularitu, tá je vhodná pri hľadaní najlepších parametrov pre jednotlivé súčasti modelu. Napríklad v predspracovaní môžeme urobiť farebnú korektúru pričom neovplyvníme nastavenie ostatných častí modelu. Taktiež v neskorších fázach môžeme pracovať s uloženým výstupom predchádzajúcich častí, čo je užitočné pri použití častí, ktoré sú časovo a zdrojovo náročné na výpočet ako napríklad extrakcia príznakov pomocou hlbokých konvolučných sietí, alebo výber príznakov, prípadne analýza hlavných komponentov. Môžeme taktiež ktorúkoľvek časť modelu zameniť za inú, napríklad použitie inej metódy pre výber príznakov, použitie iného klasifikátoru.



Obr. 3.2: Pozícia zvolených pamiatok na mape



Obr. 3.3: Návrh modelu klasifikátoru

3.4.1 Extrakcia a Selekcia príznakov

Súčasťou návrhu ako som spomenul v predchádzajúcej časti je extrakcia príznakov. Vo svojom návrhu využijem dva druhy hlbokých konvolučných sietí. V predchádzajúcej kapitole som sa zmienil o sieti *ResNet* a sieti *DenseNet*. Tieto dve hlboké neurónové siete využijem pri extrakcií príznakov, a to tak že vezmem výstup z ich posledných konvolučných vrstiev a ten prehlásim za vektor príznakov. Obe siete majú na poslednej konvolučnej vrstve vektor o 1 000 prvkoch. Keďže sa jedná o hlboké konvolučné siete je vhodné ich výpočet vykonávať na grafických kartách, ktoré poskytujú operácie pre jednotlivé vrstvy týchto sietí, čo urýchľuje výpočet rádovo v desiatkach až stovkách sekúnd.

Následne môžeme na tomto 1 000 prvkovom vektore urobiť selekciu príznakov, toto je vhodné pokiaľ chceme použiť klasifikátory, ktoré sa ťažko, prípadne dlho učia na vysoko dimenzionálnom priestore. Vzhľadom na plánované využitie neurónových sietí s výpočtom na GPU som tento krok preskočil, nakoľko môj návrh nie je určený na vysoký výkon ale len ako ukážka možností klasifikácie pamiatok.

3.4.2 Výber klasifikačného modelu

Základným modelom ktorý použijem v testoch je viacvrstvová perceptronová sieť. Táto neurónová sieť sa javí ako vhodný doplnok k extrakcií príznakov, ktorá je tiež riešená neurónovou sieťou. Výhodou je taktiež skutočnosť, že sa dá rýchlo učiť na grafických kartách, pričom naučený model je možné premiestniť a pomocou neho klasifikovať aj na procesoroch. Taktiež na neurónové siete existuje veľa frameworkov [13] [14] [15], ktoré umožňujú zostaviť tieto siete podľa vlastného uváženia s rôznymi vrstvami a aktivačnými funkciami.

Čiže základným testovaným klasifikačným modelom je viacvrstvová perceptronová sieť, ďalším typom siete bude viacvrstvová perceptronová sieť s *shortcut links* ako pri *DenseBlock*, avšak v podblokoch bude miesto konvolučných vrstiev obsahovať štandardnú plne prepojenú vrstvu, s ktorou sa bežne stretávame ako so skrytou vrstvou.

Testovanými modelmi budú aj *K-NN* a *Rozhodovacie Stromy*. Poslednými testovanými modelmi budú *ensemble* metódy *Bagging* a *Boosting* postavené nad *Rozhodovacími Stromami*.

3.5 Validácia

Výsledný model je nutné zvalidovať pomocou krížovej validácie. Krížových validácií existuje veľa druhou, príkladom *Leave-One-Out*, *Leave-P-out*, *K-fold* a ďalšie. Vo svojej práci využijem variantu *K-Fold*, takzvaný *Stratified Shuffle split*, táto metóda rozdelí podľa parametrov vstupný súbor vzoriek na dve časti, trénovaciu a testovaciu, pričom zachováva pomer vzoriek v triedach v testovacích a trénovacích súboroch. Validácia tohoto typu je vhodná napríklad na určenie minimálnej veľkosti súboru vzoriek tak aby klasifikácia bola stále dostatočne úspešná.

Základné klasifikačné dáta budú pozostávať z dát získaných z internetu, no taktiež vytvorím menšieho súbor dát nafotených mobilom a fotoaparátom. Oby dva súbory dát sú potrebné pre testovanie reálnych podmienok, kontrolu výslednej kvality modelu a celej integrácie.

Výsledkom takejto validácie by mali byť hodnoty presnosti (*Accuracy*), *Precision* a *Recall*. Nakoľko meraní jedného modelu bude viac, je vhodné tieto hodnoty spriemerovať s kladením dôrazu na ich strednú hodnotu a rozptyl. Keďže hodnoty pre *Precision* a *Recall* sú priradené priamo triedam, spriemerujeme ich najprv cez triedy a následne iterácie. Hodnoty *Precison* a *Recall* je vhodné kontrolovať aj priamo pre triedy bez spriemerovania, nakoľko môžu indikovať nevyváženosť vstupného súboru vzoriek.

3.6 Výber vhodných parametrov

Model popísaný v predchádzajúcich častiach je zložený z viacerých modulov, či už sa jedná o predspracovanie fotografie, extrakciu príznakov, samotný klasifikátor alebo diskriminačnú funkciu. Všetky tieto časti majú rôzne konfigurovateľné parametre, ktoré dokážu ovplyvniť kvalitatívne parametre celkového modelu.

3.6.1 Parametre predspracovania fotografie

Parametrami predspracovania fotografie sú spôsob zmenšenia, zväčšenia fotografie a v prípade nepomeru strán jej orezanie alebo doplnenie. Ďalšími parametrami sú napríklad parametre pri korekcií farieb, prípadné transformácie, ktoré natáčajú alebo posúvajú obraz. Bežne sa pred extrakciou príznakov stretávame s odčítavaním farebného priemeru z obrázokov. Využitie tohoto prístupu môže byť taktiež parametrom. Môj návrh však využíva základné predspracovanie, ktoré pozostáva zo zmenšenia obrázku a jeho doplnenie aby pomer strán bol spracovateľný v ďalšej fázy, ktorou je extrakcia príznakov.

3.6.2 Parametre extrakcie príznakov

Nakoľko samotná extrakcia príznakov prebieha pomocou predtrénovanej neurónovej siete, je možné meniť parametre len častí, ktoré priamo nesúvisia s uloženými váhami, napríklad môžeme upravovať typy *Pooling funkcií*, *Aktivačných funkcií* a niektoré parametre *Batch normalisation*. Taktiež je možné pridať ďalšie vrstvy aj medzi už existujú no názvy pôvodných by mali zostať rovnaké. Takáto sieť taktiež bude vyžadovať pretrénovanie pre nastavenie váh nových vrstviev.

V mojom návrhu za parameter extrakcie považujem aj výber predtrénovanej hlbokej konvolučnej siete. Pre oba mnou využívané typy sietí *ResNet* a *DenseNet*, existujú parametre, ktoré určujú ako daná sieť nakoniec vypadá a akú má hĺbku.

3.6.3 Parametre klasifikátoru

Vzhľadom na to, že mojim zámerom je otestovať viacero klasifikátorov, je samotný výber parametrom. Klasifikátory sami o sebe majú veľa parametrov, od tých najjednoduchších ako je parameter K v klasifikátore *K-NN*, ktorý určuje počet najbližších susedov podľa ktorých sa má určiť trieda, až po rozloženie neurónovej siete. Celá práca je však smerovaná k využitiu neurónových sietí, z toho dôvodu parametre ostatných metód sú pokryté len veľmi málo.

3.6.3.1 K-NN

Tento klasifikátor má dva základné parametre, prvým je parameter K, ktorý určuje počet najbližších susedov podľa ktorých sa určuje trieda. Druhým je parameter určujúci spôsob výpočtu vzdialenosti medzi jednotlivými bodmi, respektíve vzorkami.

Pri použití tohoto klasifikátoru, budem upravovať len parameter K. Ako typ vzdialenosti využívam *euklidovskú vzdialenosť*.

3.6.3.2 Rozhodovacie stromy

Rozhodovacie stromy majú viacero hlavných parametrov. Prvým je *výberové kritérium*. Toto kritérium určuje akým spôsobom sa ohodnotia parametre podľa ktorých sa bude deliť uzol. Ďalšími sú maximálna hĺbka stromu, minimálny počet vzoriek pre rozdelenie uzlu, minimálny počet vzoriek v liste, maximálny počet príznakov podľa ktorých sa bude uzol deliť a spôsob výberu súboru príznakov podľa ktorého sa bude deliť a ďalšie. Samotné rozhodovacie stromy majú veľa parametrov, ktorými sa dá vyladiť ich hĺbka, rýchlosť učenia a rýchlosť samotnej klasifikácie.

V mojej práci využijem na testovanie len výberové kritérium, spôsob výberu počtu použitých príznakov pri delení a maximálny počet príznakov pri delení. Tieto parametre priamo ovplyvňujú šírku stromu, obecne by malo platiť, že čím je strom širší, tým je plytší. Čo v konečnom dôsledku ovplyvňuje rýchlosť klasifikácie nových vzoriek.

Fixné parametre tohoto klasifikátoru neobmedzujú hĺbku stromu, minimálny počet vzoriek v liste je najmenej jeden. Minimálny počet vzoriek pre rozdelenie uzlu sú dva. Tieto fixné parametre dovoľujú aby strom rástol do hĺbky bez obmedzenia.

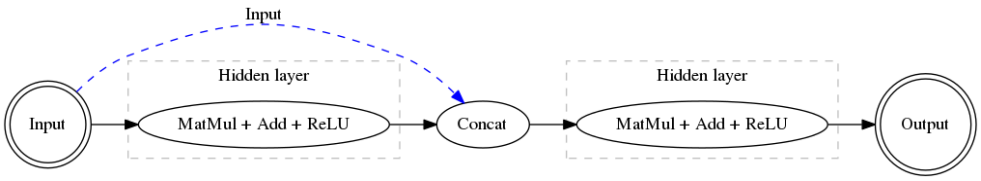
3.6.3.3 Ensemble metódy

Ensemble metódy nie sú priamo klasifikátorom, ale modelom pre metaučenie, ktorého výstupom je klasifikátor zložený z viacerých modelov. Ako som už spomenul v predchádzajúcich častiach, v práci testujem dve *Ensemble metódy*, tými sú *Bagging* a *Boosting*, pričom hlavnými parametrami týchto ensemble metód sú použitý model a počet inštancií, ktoré trénuje.

Počas testovania ako model fixne použijem *Rozhodovacie stromy*, počet inštancií bude parameter ktorý budem testovať.

3.6.3.4 Viacvrstvové perceptrony

Viacvrstvové perceptrony (MLP), sú samostatnou kategóriou klasifikátorov, nakoľko samotná štruktúra siete je parametrom. Štruktúra je definovaná jednotlivými vrstvami. Za vrstvy považujeme násobiace vrstvy, sčítacie vrstvy, vrstvy aktivačných funkcií, vrstvy doplnkových funkcií ako *Dropout* alebo



Obr. 3.4: Grafické znázornenie viacvrstvového perceptronu s *DenseBlock* veľkosti 1

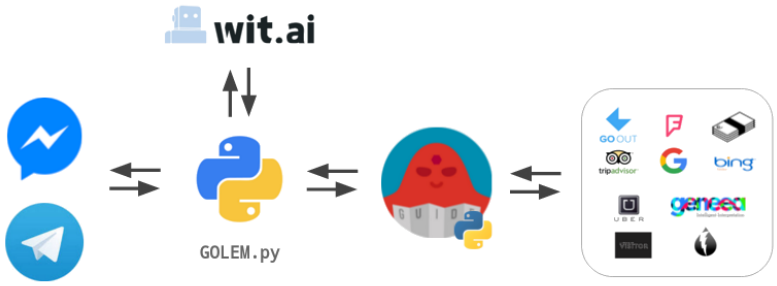
Concatenation. Myšlienka rozdelenia na vrstvy je prevzatá z frameworkov [13] [14], ktoré slúžia k modelovaniu neurónových sietí a ich učeniu. Ďalšími parametrami sú parametre samotného učenia ako použitý algoritmus na optimalizáciu váh, rýchlosť učenia (*Learning rate*), postup rýchlosti učenia (*Learning rate policy*), veľkosť dávky (*Batch size*), maximálny počet epoch (*Maximal iteration count*), použitie momentu (*Momentum*) a jeho váha. Obecne parametrov neurónových sietí je veľa a samotná skladba siete a použité doplnkové modifikácie algoritmu ich razantne upravuje.

Mnou testované parametre v tomto prípade budú štruktúra siete a maximálny počet iterácií, pričom ako pre optimalizáciu použíjem *Adam Optimalizátor* [16]. Rýchlosť učenia bude konštantná a nastavená na 0.001. V testoch využíjem dva druhy sietí. Prvou je štandardný troj vrstvový perceptron s jednou skrytou vrstvou, kde parametrami sú počet uzlov v skrytej vrstve a aktivačná funkcia.

Druhým typom siete bude perceptron s jedným *DenseBlock*, ktorého veľkosť (počet podvrstiev), počet uzlov v podvrstvách a aktivačná funkcia medzi podvrstvami sú parametrami. V tomto prípade budú veľkosti všetkých podvrstiev rovnaké a taktiež aktivačná funkcia. Tento typ siete je znázornený na obrázku 3.4. Vrstva *Concat* pripája k výstupnému vektoru prvej skrytej vrstvy pôvodný vstup, tento vektor je následne poslaný druhej skrytej vrstve.

3.6.4 Parametre Validácie

Ako som už spomenul v predchádzajúcej časti, parametrami validácie sú počet iterácií a pomer medzi testovacím a trénovacím súborom vzoriek. Oba parametre počas testovania fixne nastavím. Pomer medzi trénovacím a testovacím súborom vzoriek považujem za vhodné nastaviť na 0.5. Toto nastavenie spôsobí, že veľkosti trénovacej a testovacej množiny budú rovnaké. Druhým nastavovaným parametrom je počet iterácií, tento nastavím na 5, táto hodnota by mala byť dostačujúca pre účely validácie, vyššie hodnoty by ju zbytočne predĺžili, pričom výsledné spriemerované hodnoty validácie by zostávali sa razantne nezmenili.



Obr. 3.5: Návrh Prague Visitor chatbota postaveného na *Golem* frameworku

3.7 Chatbot a Golem Framework

Chatbot je aplikácia, ktorá slúži na interakciu s užívateľom pomocou ľudom prirodzeného spôsobu a to určitou formou konverzácie. Tento typ botov sa v poslednej dobe zviditeľnil primárne vďaka súťaži *Alexa Challenge* od *Amazon*, ktorej cieľom bolo vytvoriť bota schopného interagovať s ľudmi čo po najdlhší čas bez toho aby sa opakoval, prípadne zacyklil. Momentálne sa dopyt po chatbotoch rozrastá, Ich uplatnením môžu byť napríklad palubné počítače v autách, lietalách a iných dopravných prostriedkoch, ako aj bežné využitie ako odborného asistenta, prípadne inteligentného nástroja pre užívateľský vstup.

Golem je framework vyvíjaný u nás na Fakulte informačních technologií študentmi. Vývoj započal ako projekt pre spoločnosť *Prague Visitor* cez *Portál Spolupráce s Priemyslom*. Je písaný v jazyku *Python* verzie 3, dokáže pracovať so správami z aplikácií *Facebook Messanger* a *Telegram*. Primárny návrh bol zameraný na vývoj frameworku pre chatbotov, pričom nad týmto frameworkom bol neskôr postavený chatbot práve pre doporučovanie akcií v Prahe. Celý koncept je postavený na konečnom stavovom automate, pričom užívateľ interakciou prechádza medzi stavmi. Výhody takéhoto Chatbota je jednoduchosť a rýchlosť, nakoľko pre určenie ďalšieho stavu nie je potrebný žiadny výpočet. Návrh celého chatbota nájdeme na obrázku 3.5. Z ľava si môžeme všimnúť priebeh komunikácie, *Golem* framework najprv príjme správu z *Facebook Messanger* alebo *Telegram*, tú si prevedie do internej reprezentácie, následne jej obsah spracuje pomocou *wit.ai*, čo je online služba slúžiaca na spracovanie prirodzeného jazyka (*NLP*). Táto služba vráti takzvaný *intent*, čo je akoby jadro, prípadne význam správy, či už je to pozdrav alebo otázka. Framework tento *intent* spracuje a na základe aktuálneho kontextu (momentálne len aktuálny stav) vykoná akciu preddefinovanú konečným stavovým automatom.

Samotné správy sú v *JSON* formáte a skladajú sa z dvoch častí. Prvou je čistý text, inak povedané text, ktorý užívateľ pošle druhej strane. Ďalšou častou sú prílohy, tieto prílohy môžu byť:

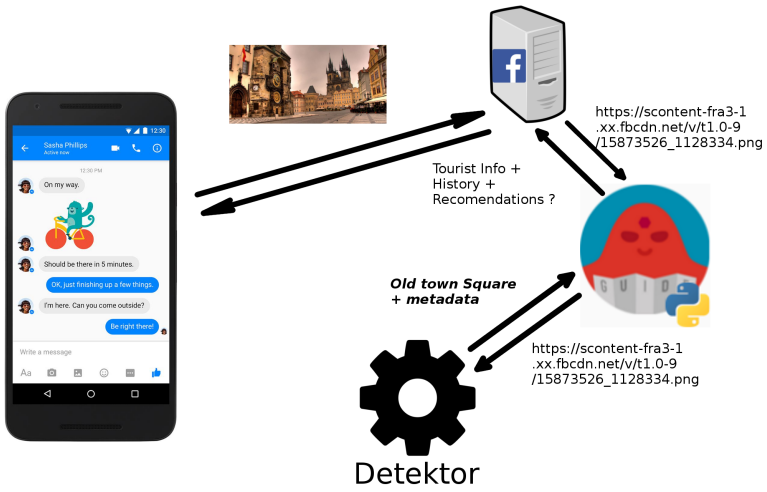
- Súbory - obecné súbory iného charakteru než audio, obrázky a video
- URL adresy
- Audio
- Obrázok
- Video
- GPS pozícia

Audio, video a obrázky sú v prípade *Facebook Messanger* posielané ako odkaz na jeden z ich úložných serverov. Tento prístup zaručuje malú veľkosť správ čo je vhodné v prípade, že framework ktorý dané správy spracováva a nevyužíva prílohy. *Golem* framework ignoruje obrázkové prílohy, no ich spracovanie je možné jednoducho doimplementovať vďaka vhodne použitému návrhu. Rozšírením by bolo vytvorenie špeciálneho stavu, ktorý by spracovával tieto prílohy za použitia aktuálneho kontextu s tým, že by sa ignoroval stav, v ktorom sa daný stavový automat nachádzal predtým. Tento prístup sa dá prirovnať k spracovaniu prerušení v hardware-ových prostriedkoch.

3.8 Integrácia

Základný návrh toku dát môžeme vidieť na obrázku 3.6. Z ľava, užívateľ odošle fotografiu pomocou aplikácie *Facebook Messanger* na server spoločnosti *Facebook*, tá danú správu spracuje a odošle ju na *Golem* framework, ktorý správu prečíta. Keď zistí že obsahuje fotografiu, odošle ju na detektor, tento detektor podľa návrhu modelu klasifikátora zobrazeného na obrázku 3.3, spracuje a vráti názov. Názov je jedna z vybraných historických pamiatok zo zoznamu 3.1. Framework následne informuje užívateľa naspäť o tom na akú pamiatku sa práve pozerá, prípadne ho informuje o tom, že daný objekt na fotografií, nie je rozpoznaná pamiatka. Samotný *detektor*, ako som v tomto prípade backend na rozpoznávanie nazval. Detektor je jednoduchá aplikácia na základe klient-server architektúry. Detektor by mal príjmať požiadavky na spracovanie obrázku, či už bude obrázok predaný ako odkaz na umiestnenie na internete alebo zakódovaný ako *Base64* reťazec.

Ďalšou doplnkovou funkciou serveru by mala byť schopnosť vykonať krížovú validáciu a vrátiť jej dáta. Táto vlastnosť je vhodná pre ukážku a zhodnotenie úspešnosti zvoleného modelu. Poslednou doplnkovou funkciou by mala byť schopnosť vrátiť informácie o vstupných dátach na ktorých bol model naučený. Táto funkcia je čisto informatívna. Tieto doplnkové funkcie by nemali byť priamo dostupné pomocou *Facebook Messanger*, ale ako rozšírenie cez jednoduché kontrolné *API*.



Obr. 3.6: Návrh integrácie klasifikátoru pamiatok do *Golem* frameworku

Kapitola 4

Realizácia

V tejto kapitole sa zameriam na celkovú realizáciu. Dlo sťahovania jednotlivých fotografií, následnú filtráciu, predspracovanie, extrakciu príznakov, učenie klasifikačných modelov až po validáciu a hľadanie najvhodnejšieho modelu. Počas realizácie som využil rôzne nástroje, primárne však vlastné programy písané v jazyku python. Na prácu s neurónovými sieťami som využil frameworky *Caffe* [14], *TensorFlow* [13] a *Scikit Learn (sklearn)* [15]. Ostatné klasifikačné modely som využil z balíku *Scikit Learn (sklearn)* [15], tento balík obsahuje implementáciu všetkých potrebných klasifikačných modelov a taktiež nástroje pre spracovanie dát ako napríklad nástroje pre redukciu dimenzionality (*PCA*, výber príznakov). Ďalšími použitými balíkmi sú *Numpy*, čo je knižnica pre rýchle matematické výpočty pre *Python*. Rýchlosť je zabezpečená použitím dátových štruktúr a výpočtov v C-čkovom prostredí miesto využitia matematických operácií priamo v jadre python, ďalej *OpenCV*. Táto knižnica slúži na manipuláciu s obrázkami a videami. Obsahuje široké množstvo funkcií pre spracovanie obrazu, filtrovanie, manipuláciu a ďalšie ako napríklad detekciu hrán, implementáciu algoritmov ako *SIFT* a *SURF* a iné. Ostatnými nástrojmi použitými v tejto práci sú *jupyter*, *curl*, *parallel* [17]. Samotné testovanie prebiehalo na výpočtovom klustry organizácie *Metacentrum*, kde som využil dostupné GPU uzly pre testovanie neurónových sietí a extrakciu príznakov pomocou predtrénovaných hlbokých konvolučných sietí. Celkovo som vytvoril 12 209 úloh, ktoré celkovo trvali 1 300 CPU dní.

4.1 Zber dát

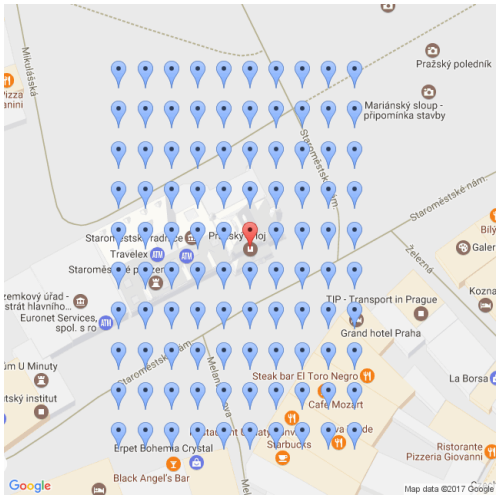
Prvým krokom realizácie bol zber dát, v našom prípade stiahnutie relevantných obrázkov z internetu, ako som spomenul v predchádzajúcej kapitole, máme tri druhy zdrojov. Prvým je *Google Street View* ktorého panoramatické fotky sú dobre dostupné pri pamiatkach, okolo ktorých sa nachádzajú cesty alebo sú dostatočne známe a je k nim prístup na bicykli. Druhým zdrojom je *Google Custom Search*, ktorý je známejší po názvom *Google Image Search*,

tento obsahuje kvalitné fotky pamiatok, ktoré sú obecne známe. Posledný zdrojom sú fotografií su snímky extrahované z videa, ktoré som natočil pri vybraných pamiatkach. Ako prvý zdroj popíšem *Google Street View*, nakoľko som prácou s týmto zdrojom strávil najviac času. Na stiahnutie dát z internetu som využil vlastné skripty písane v jazyku *Python*. Na uloženie dát pre postupné spracovanie som využil *SQLite3* databázu.

Celkovo bolo získaných 118 406 fotografií z čoho 100 465 bolo zo snímkov z videa a 17 941 internetu. Fotografie stiahnuté z internetu boli neskôr filtrované.

4.1.1 Google Street View

Nakoľko pre získanie fotografie bolo potrebné zadať zemepisné súradnice (zemepisnú výšku a šírku) a taktiež uhol natočenia kamery, bolo prvým krokom geokódovanie zvolených pamiatok na zemepisnú výšku a šírku. To som docielil pomocou služby *Google Geocoding*, ktorá poskytuje jednoduché *API*, v ktorom stačí zadať len textový popis objektu, prípadne jeho adresu. Akonáhle máme zemepisné súradnice jednotlivých pamiatok, vytvoríme okolo nich mriežku (príklad je na obrázku 4.1). Pre každý uzol v mriežke, získame metadáta, z ktorých overíme, či na daných súradniciach existuje fotografia, pričom si ukladáme vrátené koordináty pre neskoršie spracovanie. Navrhnutá mriežka má uzly od seba vzdialené 2 metre a celkovo má 100 uzlov v x-ovej a v y-ovej osy, z čoho vyplýva, že najväčšia vzdialenosť, z ktorej je pamiatka vyfotografovaná približne 142 metrov. Tieto hodnoty boli zistené empiricky, vzhľadom na vzdialenosti budov v centre mesta, kde sa táto hodnota ukázala za hraničnú.



Obr. 4.1: Ukážka mriežky okolo vybranej pamiatky

Po dokončení prvej fázy nasleduje výpočet základného uhlu medzi pozíciou z ktorej chceme získať fotku voči pozícií prislúšnej pamiatky. Vzťah pre tento výpočet tohto uhlu je na obrázku 4.2. Následne vygenerujeme ďalšie štyri

$$\begin{aligned}\phi, \delta &= latitude, longtitude \\ X &= \delta_2 - \delta_1 \\ Y &= \log\tan\left(\frac{\phi_2}{2} + \frac{\pi}{4}\right) - \log\tan\left(\frac{\phi_1}{2} + \frac{\pi}{4}\right) \\ heading &= |atan2(X, Y) + 360|_{360}\end{aligned}$$

Obr. 4.2: Vzťah pre výpočet uhlu medzi dvoma bodmi zadanými koordinátami

Obr. 4.3: Ukážka pozíc fotografií pre vybranú pamiatku

pozície pri pamiatke vo vzdialenostiach -10, -5, 5 a 10 metrov od základnej pozície v uhle kolmom na uhol, pod ktorým budeme získavať fotografie. Názorná ukážka je na obrázku 4.4, kde modrý bod je základná pozícia pamiatky, červený bod je miesto, na ktorom bola fotografia vytvorená a zelené body sú pozície, vzhľadom na ktoré počítame tieto uhly.

Vďaka tomuto rozšíreniu zvýšime počet fotografií niekoľko násobne. Posledným rozšírením súboru fotografií je pridanie uhlu v y-ovej osi, nakoľko jedna fotografia nemusí obsiahnuť celú pamiatku, pokiaľ je fotená v jej tesnej blízkosti. Sťahujeme fotografie, ktoré sú pod uhlami 0 °, 15 °, 30 °a 45 °. Týmto spôsobom pokryjeme pamiatku aj pokiaľ je fotografia fotená v tesnej blízkosti. Poslednou fázou je samotné sťahovanie fotografií, ktoré prebieha postupne vo fronte.

Týmto spôsobom sa podarilo získať 13 460 fotografií z celkových 47500 testovaných, ktoré bolo potrebné ďalej spracovať, presné počty sú v tabuľke 4.1.

Obr. 4.4: Ukážka pozíc a rozloženia prídavných pozíc pre uhly fotografie na danej pozícií a pre vybranú pamiatku

Name	Count
Basilica of St. Peter and St. Paul	1 040
Brick gate	1 056
Church of Our Lady before týn	12 25
Clementinum	1 088
Dancing House	880
Emauss	272
Lennons Wall	240
Leopold Gate	1 600
National Theatre	1 375
Prague Astronomical clock	2 635
Rudolfinum	806
Strahov Monastery	240
The Bethlehem Chapel	432
The Estates Theatre	475
The Municipal House	560
The Powder Tower	576
Total	13 460

Tabuľka 4.1: Výčet počtov získaných fotografií z *Google Street View*

Basilica of St. Peter and St. Paul	60
Brick gate	60
Church of Our Lady before týn	64
Clementinum	59
Dancing House	60
Emauss	73
Lenons Wall	61
Leopold Gate	58
National Theatre	58
Prague Astronomical clock	62
Rudolfinum	59
Strahov Monastery	60
The Bethlehem Chapel	60
The Estates Theatre	59
The Municipal House	60
The Powder Tower	60
Total	973

Tabuľka 4.2: Výčet počtov získaných fotografií z *Google Custom Search*

4.1.2 Google Custom Search

Ďalším zdrojom fotografií je *Google Custom Search*. Pre zvolený súbor pamiatok som sa rozhodol stiahnuť 50 stránok, pre každú pamiatku, čo sa ukázalo ako vhodné, nakoľko vyhľadávanie vracia veľa krát aj rovnaké odkazy len z iných zdrojov. Týmto prístupom sa podarilo celkovo získať 973 fotografií z celkovo testovaných 950 stránok, teda 9 500 testovaných odkazov. Výčet presných počtov je v tabuľke 4.2.

4.1.3 Vlastné fotografie

Posledným zdrojom fotografií sú jednotlivé snímky z osobne natočených videí vybraných pamiatok. Videá som sa rozhodol ísť nakrúcať z rána okolo siedmej hodiny rannej, nakoľko v tom čase sa v meste nachádza ešte málo turistov a okolo pamiatok, ako napríklad *Pražské astronomické hodiny*, ktoré sú najnavštevovanejšou historickou budovou v Prahe sa dá pohybovať bez problémov. Pamiatky som sa rozhodol nakamerovať svojím telefónom, keďže výsledný model má byť otestovaný krížovou validáciou práve na fotografiách z mobilného telefónu. Použil som telefón *Huawei P8 lite*. Fotografie a video je schopný vytvoriť vo Full HD rozlíšení teda 1920x1080 pixlov, pričom video nakrúca pri maximálnej snímkovacej frekvencií 30 snímkov za sekundu. Každá pamiatka bola nakrútená niekoľko krát, z rôznych uhlov, pričom pri nakrúcaní som kamerou pomaly zachytával celý objekt tak, aby vo výslednom videu

Basilica of St. Peter and St. Paul	7 617
Brick gate	5 243
Church of Our Lady before týn	3 496
Clementinum	6 890
Dancing House	3 376
Emauss	6 419
Lennons Wall	5 434
Leopold Gate	6 982
National Theatre	6 839
Prague Astronomical clock	5 246
Rudolfinum	8 576
Strahov Monastery	5 972
The Bethlehem Chapel	6 557
The Estates Theatre	9 062
The Municipal House	5 601
The Powder Tower	7 155
Total	100 465

Tabuľka 4.3: Výčet počtov získaných fotografií z osobne natočených videí

bol niekoľko krát zobrazený v rôznych pozíciach na snímku. Každé video má približne jednu minútu a každá pamiatka bola nakrúcaná minimálne z troch rôznych pozíc. Týmto spôsobom vzniklo celkovo 65 videí, z ktorých som následne pomocou nástroja *ffmpeg* vyextrahoval jednotlivé snímky, čím vzniklo celkovo 100 465 fotografií. Celkový zoznam sa nachádza v tabuľke 4.3.

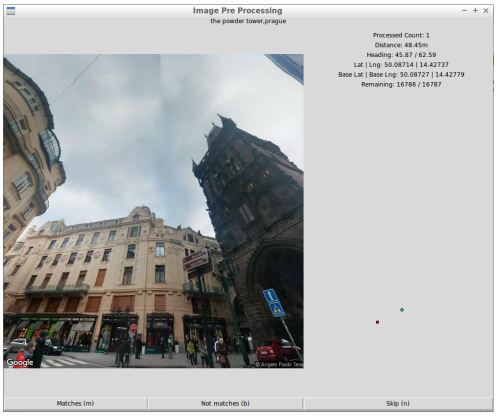
4.2 Filtrácia

Ďalším krokom po získaní dát bola filtrácia nevhodných fotografií. Za nevhodnú fotografiu považujem fotografiu, ktorá neobsahuje zvolenú pamiatku, zvolená pamiatka je na fotografií nejasne vidno, pamiatka je na fotografií poškodená alebo sa jedná o fotografiu interiéru pamiatky. Príklady uvádzam na obrázku 4.5.

Vzhľadom na množstvo fotografií som sa rozhodol vytvoriť jednoduchú aplikáciu, na ich filtrovanie. Písaná je v jazyku *Python* a poskytuje jednoduché užívateľské rozhranie pre určenie, či obrázok sa zhoduje s požiadavkami alebo nie. Aplikácia ponúka možnosť preskočiť na ďalší obrázok a taktiež zobrazuje informácie o obrázku, ako vzdialenosť od pamiatky, uhol smerom k pamiatke, uhol v y-ovej osy ako aj grafické znázornenie, kde sa pamiatka nachádza vzhľadom na GPS súradnice. Tieto informácie sú užitočné pre hľadanie chýb a kontrolu, či obrázok zodpovedá dátam s ktorými bol získaný, nakoľko sa môže stať, že získané dáta obsahovali zlé metadáta, čo mohlo byť spôsobené

Obr. 4.5: Ukážka nevhodných fotografií, z ľava: chýbajúca pamiatka, pamiatka nie je dobre rozpoznateľná, poškodený snímok a fotografia interiéru

zlou kalibráciou fotoaparátu.



Obr. 4.6: Ukážka aplikácie na odstránenie nevhodných fotografií

Po filtrácií celkovo zostalo 6 741 vhodných fotografií, z čoho 6 177 je z *Google Street View* a *564* z *Google Custom Search*. Počet nevalídných fotografií z *Google Street View* prisudzujem tomu, že veľa ich bolo z interiéru danej budovy, no ešte viac ich neobsahovalo pôvodnú pamiatku. Nakoľko mnou zvolená mriežka bola dostatočné veľká aby pokrila vedľajšie ulice, alebo sa jednalo o fotografiu z interiéru budov okolo ako sú napríklad obchody. Celkový zoznam fotografií po filtrácií sa nachádza v tabuľke 4.4.

4.3 Predspracovanie

V návrhu som spomenul, že predspracovanie v mojej realizácií pozostáva zo zmenšenia, respektíve zväčšenia obrázku na vhodnú veľkosť, v tomto prípade 224x224 pixlov. Pokiaľ obrázok nie je vo vhodnom pomere strán, je najprv vytvorený výrez tak, že sa zoberie kratšia strana a pomocou nej sa vytvorí

Basilica of St. Peter and St. Paul	639
Brick gate	438
Church of Our Lady before týn	709
Clementinum	95
Dancing House	557
Emauss	206
Lennons Wall	110
Leopold Gate	424
National Theatre	543
Prague Astronomical clock	1524
Rudolfinum	312
Strahov Monastery	156
The Bethlehem Chapel	159
The Estates Theatre	411
The Municipal House	130
The Powder Tower	328
Total	6741

Tabuľka 4.4: Výčet počtov fotografií po filtrácií

štvorec so stredom v strede pôvodného obrázku. Druhou možnosťou je rozšírenie pôvodného obrázku na štvorec a nastavenie nových pixlov na farebný priemer pôvodného obrázku.

Nakoľko sú vstupné súbory obrázkov relatívne malé (až na nafotené fotografie), rozhodol som sa vytvoriť z pôvodných fotografií určité nové fotografie pomocou rôznych efektov. Z efektov som si vybral rozmazanie fotografie, zaostrenie, rotáciu a zmenu jasu. Tieto efekty sú zvolené tak, aby imitovali rôzne javy, ktoré sa môžu pri fotografovaní vyskytnúť a tak, aby nové súbory obrázkov pokryli viac možných prípadov.

4.3.1 Rozmazanie fotografie

Rozmazanie fotografie prebieha pomocou štandardného *Gausovského kernelu* o veľkostiach 3x3 (ukážková je vo vzťahu 4.7), 9x9 a 17x17. Aplikácia prebieha postupným váženým sčítavaním hodnôt farieb okolitých pixlov podľa dopredu daného kernelu. Na výpočet tejto matice a jej aplikáciu na fotografiu som použil metódu *GaussianBlur* z knižnice *OpenCV* pre *Python*.

4.3.2 Zaostrenie fotografie

Na zaostrenie fotografie využívam techniku nazývanú *Digital unsharp masking*, kde sa od pôvodnej fotografie odčítava táto istá fotografia, ale rozma-

$$\frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Obr. 4.7: Ukážka Gausovského kernelu o veľkosti 3x3

$$Sharpened = Original + (Original - Blurred) \cdot Amount$$

Obr. 4.8: Vzťah pre výpočet zaostrenia fotografie

$$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \begin{bmatrix} cos\ \theta & sin\ \theta \\ -sin\ \theta & cos\ \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Obr. 4.9: Ukážka bežnej rotačnej matice

zaná. Tento proces je vyjadriteľný vzťahom ktorý je uvedený na obrázku 4.8. Hodnota *Amount* a parametre rozmazania sú parametrami tejto techniky.

Oba parametre v predspracovaní nastavujem na rovnakú hodnotu. Ako tieto hodnoty som zvolil 8, 6.5, 5. Hodnoty som vybral po predchádzajúcom testovaní rôznych hodnôt.

4.3.3 Rotácie

Na otočenie obrázku sa používa rotačná matica (viď obrázok 4.9), ktorá nám určuje nové súradnice jednotlivých pixlov v závislosti na uhle natočenia. Tie som zvolil následovne: -22 °, -13.2 °, -4.4 °, 4.4 °, 13.2 °a 22 °, tieto uhly sú volené tak, aby pokryli určité možné "chybné"natočenie fotografie a taktiež pre testovanie väčšieho natočenia.

4.3.4 Zmena jasu

Zmena jasu je posledným efektom, ktorý aplikujem na obrázky. Týmto efektom sa snažím simulovať svetelné zmeny, ktoré nastávajú počas dňa a taktiež počas celého roka. Tento efekt je najjednoduchším, nakoľko ho riešim pričítaním konštanty ku všetkým hodnotám farieb. Tieto hodnoty sú -100, -60, -20, 20, 60, 100. Voľba týchto hodnôt je založená podobne ako v predchádzajúcej časti na experimente.

4.4 Extrakcia Príznakov

Po tomto predspracovaní je obrázok poslaný cez predtrénovanú hlbokú konvolučnú sieť, ktorej výstupné vrstvy sú ďalej použité ako príznaky. Ako som

spomenul v návrhu, pre potreby extrakcie príznakov využijem hlboké konvolučné siete *ResNet* [11] a *DenseNet* [12], výstupom oboch sietí je vektor o 1000 prvkoch. Tieto predtrénované modely sú vytvorené pomocou frameworku *Caffe* [14] a sú verejne dostupné na kolaboračnom nástroji *Github*, na adresách `https://github.com/KaimingHe/deep-residual-networks` a `https://github.com/shicai/DenseNet-Caffe` spolu s váhami predtrénovanými na *ImageNet dataset*.

Testované hlboké konvolučné siete v tomto prípade boli *ResNet-50*, *ResNet-101*, *ResNet-152*, *DenseNet-121*, *DenseNet-161*, *DenseNet-169* a *DenseNet-201*, kde číslo za názvom siete označuje hĺbku siete. Extrakcia príznakov prebiehala na GPU klustry organizácie *MetaCentrum*, nakoľko sa jedná o hlboké siete, ktoré obsahujú veľké množstvo váh. Pre všetky zvolené siete som vytvoril dataset obsahujúci vektory príznakov a zaradenie do tried jednotlivých vzoriek, s týmito súbormi pracujem v ďalších častiach modelu.

4.4.1 ImageNet dataset

ImageNet dataset sa skladá z 516 840 obrázkov, ktoré obsahujú celkovo 200 tried. Samotné obrázky môžu obsahovať viacero objektov. Celkovo sa ich nachádza na obrázkoch 534 309, čo je o 17 469 viac než je samotných obrázkov. V trénovacej množine obrázkov sa nachádza 456 567 obrázkov, na ktorých je 478 807 objektov. Validačná množina obrázkov obsahuje 20 121 fotografií a 55 502 objektov, testovacie množina pozostáva z 40 152 obrázkov pričom počet objektov nie je zverejnený.

Objekty pozostávajú z tried ako lietadlo, vták, sekera, autobus a iné, celkovo množina obrázkov pozostáva z rôznorodých objektov, živých i neživých.

4.5 Testovanie modelov

V tejto sekcií popíšem implementáciu zvolených klasifikačných modelov, priebeh ich trénovania a testovania a taktiež výsledky, ktoré dosiahli. V analýze som ako základné klasifikátory zvolil, *K-NN*, *Rozhodovacie Stromy*, *Viacvrstvové perceptrony* a ensemble metódy *Bagging* a *Boosting*. Taktiež som sa rozhodol skúsiť naučiť hlboké konvolučné siete zo získaných fotografií pre porovnanie s modelom využívajúcim predtrénované konvolučné siete na extrakciu príznakov.

Testovanie modelov prebiehalo na niekoľkých rôznych vstupných súboroch, ktoré vznikli pri predspracovaní. Tými sú jednak pôvodné fotografie získané či už z *Google Street View*, *Google Custom Search* alebo extrakciou z videa, ich deriváty, v ktorých bol upravený jas, rotácia, zaostrenie prípadne rozmazanie. A následne príznaky vyextrahované z týchto pôvodných fotografií pomocou hlbokých konvolučných sietí *ResNet* a *DenseNet* s rôznymi konfiguráciami hĺbky.

Celkovo teda testy prebiehali na 425 CSV súboroch rozdelených podľa typu siete použitej k extrakcií príznakov, zdroja pôvodných fotografií a použitého predspracovania. Tieto obsahujú 55 576 716 záznamov na jeden typ siete čo celkovo dáva 389 037 012 záznamov. Vzhľadom na dimenzionalitu extrahovaných príznakov a celkový počet záznamov som počas testov využil len zlomok záznamov, nakoľko trénovanie a klasifikácia niektorých modelov bola časovo veľmi náročná. Testy hlbokých konvolučných sietí prebiehali na pôvodných obrázkoch získaných extrakciou z videa a z *Google Street View*. Deriváty fotografií som v tomto prípade nepoužil nakoľko prvý súbor obrázkov ma 100 465 fotografií a teda jeden cyklus trénovania trval približne pol hodinu, druhý súbor (z *Google Street View*) mal naopak 7 678 fotografií no trénovanie na nich sa nepreukázalo za úspešné.

Testovanie modelov prebiehalo výhradne na serveroch virtuálnej organizácie *MetaCentrum*, ktorá poskytuje pre akademickú obec výpočtový grid, v ktorom sa nachádzajú aj uzly podporujúce výpočty na grafických kartách pomocou technológie CUDA. Celkovo prebehlo niekoľko stoviek testov s rôznymi vstupnými súbormi, na výpočet konvolučných sietí a viacvrstvových perceptronov som použil framework *TensorFlow*, ktorý tieto vrstvy priamo podporuje a umožňuje ich počítať na grafických kartách. Umožňuje načítavať dáta zo vstupných súborov ako obrázky a CSV pomocou optimalizovaných viacvláknových front priamo do vytvoreného modelu.

Na testovanie ostatných modelov ako *K-NN*, *Rozhodovacie stromy* a ensemble metódy som použil *scikit-learn* framework, ktorý síce tieto modely počíta na procesore, no dokáže ich počítať viac vláknovo a je implementovaný tak, aby nebol závislí na žiadnych hardware-ových požiadavkách. Čo sa týka týchto modelov, trénovanie a klasifikácia dokáže byť časovo náročná pri vyššej dimenzionalite prípadne vyššom počte záznamov, preto som na týchto modeloch robil testy len s pomocou základných súborov príznakov.

Pri validácií modelov som využil *Stratified K-Fold*, ktorý rozdeľuje vstupný súbor príznakov (prípadne obrázkov) na K skupín, ktoré majú rovnaké percentuálne zastúpenie jednotlivých tried. Následne sa model trénuje na $K - 1$ skupinách a posledná skupina je použitá na testovanie modelu. Merané výsledné hodnoty sú v mojom prípade *Accuracy*, *Precision* a *Recall*, pričom sa snažíme maximalizovať všetky tri.

Pre ukážku uvedádzam grafy a tabuľky obsahujúce výsledky meraní pre dátové súbory podľa typu vstupných obrázkov, typ použitej siete na extrahovanie príznakov je vo všetkých prípadoch *DenseNet-121*.

4.5.1 Základná implementácia testov

Implementované testy sa dajú zaradiť do kategórií podľa typu vstupných dát, či už sú to obrázky alebo extrahované príznaky, a potom použitého frameworku. Všetky kategórie však majú jednu spoločnú vlastnosť, využívajú na konfiguráciu premenné prostredia, pomocou tých nastavujem vstupný dátový

súbor a parametre testov. Samotné skripty teda v úvode obsahujú kontrolu, či sú požadované premenné nastavené a či nastavené parametre sú valídne.

Testy využívajúce framework *scikit-learn* sa skladajú z jednoduchého načítania CSV súboru, pričom sa z neho odstráni hlavička, indexový stĺpec a nepoužívané stĺpce. Ako napríklad názov súboru, ktorému náležia dané príznaky. Následne prichádza na radu hlavný cyklus, v ktorom sa iteruje cez testované parametre algoritmu, cyklus teda obsahuje volanie testovacej funkcie a následné uloženie výsledku, výsledok obsahuje merané parametre, namerané metriky všetkých behov validácie pre dané parametre, spriemerované metriky a nakoniec výsledky klasifikácie spojené s pravými hodnotami testovaných vzoriek. Opäť cez všetky behy validácie. Táto štruktúra výsledkov umožňuje overenie nameraných metrík, prípadne výpočet ďalších zo získaných dát. Samotná testovacia funkcia je jednoduchá, obsahuje jednoduchý cyklus, ktorý pomocou triedy *StratifiedKFold* rozdeľuje vstupný súbor príznakov na trénovaciu a testovaciu množinu. V každej iterácií cyklu sa zo vstupného súboru rozdeleného na trénovaciu a testovaciu množinu, naučí zvolený model na trénovacej množine a následne sa klasifikuje testovacia množina. Z výsledkov klasifikácie a správnych hodnôt sa vypočítajú metriky ako *Accuracy*, *Precision* a *Recall*, ktoré sa uložia pre neskoršie spriemerovanie. Táto funkcia vracia spriemerované hodnoty a všetky namerané hodnoty a metriky. Výstup týchto testov je ukladaný vo formáte *JSON*, kvôli jednoduchej štruktúre, možnosti komplexnejších dátových štruktúr a prenositeľnosti.

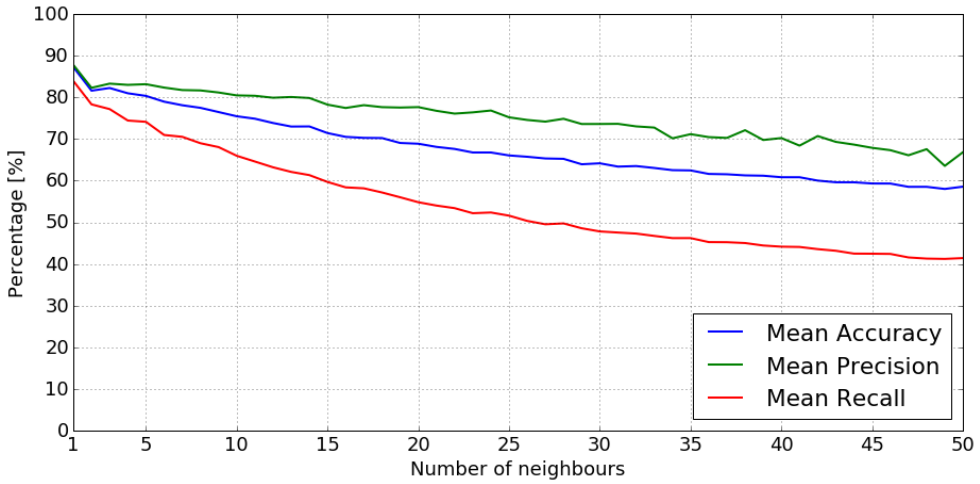
Testy využívajúce framework *TensorFlow* sú štrukturované trocha rozdielne, nakoľko obsahujú samotnú definíciu trénovanej siete a trénovací cyklus. Základom týchto testov je teda funkcia, ktorá vytvára štruktúru samotnej siete, pridáva jej vrstvy, trénované váhy a optimalizátor váh. Jej návratovými hodnotami sú teda vstupy, výstupy, optimalizátor a sledované premenné. Druhá hlavná funkcia má na starosti meranie a testovanie. Ako prvé je potreba zakódovať vstupné triedy nakoľko *TensorFlow* nedokáže pracovať s textovými triedami a pre viactriednu klasifikáciu vyžaduje zakódovanie tried pomocou *1-z-N* kódovania. Ďalej rozdelím vstupný dátový súbor na trénovaciu a testovaciu množinu, potom vytvorím fronty pre vstup. Tieto slúžia na urýchlenie výpočtu paralelným načítaním vstupných dát do internej reprezentácie *TensorFlow*. Následne sa zavolá funkcia na vytvorenie štruktúry siete. Ďalším krokom je inicializácia siete, prípadne načítanie uloženého modelu, vytvorenie triedy slúžiacej pre ukladanie záznamov do *EventLog* pre následnú analýzu pomocou nástroja *TensorBoard*. Potom nasleduje samotný trénovací cyklus, ktorý pozostáva z trénovacej fázy a testovacej fáz. V trénovacej fáze je v postupne v dávkach trénovaná sieť (*Batch training*), pričom si ukladám sledované parametre modelu ako *Chyba*, *Priemerný gradient* a *Priemerná hodnota váh*. Vývoj týchto parametrov môžem neskôr sledovať pomocou nástoja *TensorBoard*. Sú vhodné pre určenie vývoja učenia a zistenie, či model je schopný sa ešte učiť alebo sa učenie dostalo do lokálneho optima. Po trénovacej fáze prichádza na radu testovacia. Túto robím každých 10 epoch (cyklov učenia). Výsledkom

testovacej fázy sú opäť metriky *Accuracy*, *Precision* a *Recall*, ktorých hodnoty opäť ukladáme pre neskoršie zobrazenie v *TensorBoard*. Na konci testovacej fáze uložím model, pre možnosť neskoršej obnovy váh. Koniec trénovacieho cyklu obsahuje samotné pridanie nameraných dát do *EventLog*. Namerané metriky je možné získať z nástroja *TensorBoard*, pričom pomocou tohoto nástroja je možné pozerať aj vývoj týchto metrík, keďže dáta sú zo súborov čítané priebežne.

Obe základné implementácie sú navrhnuté tak, aby bolo možné jednoducho zameniť použitý model, štruktúru siete alebo vstupný súbor príznakov prípadne obrázkov.

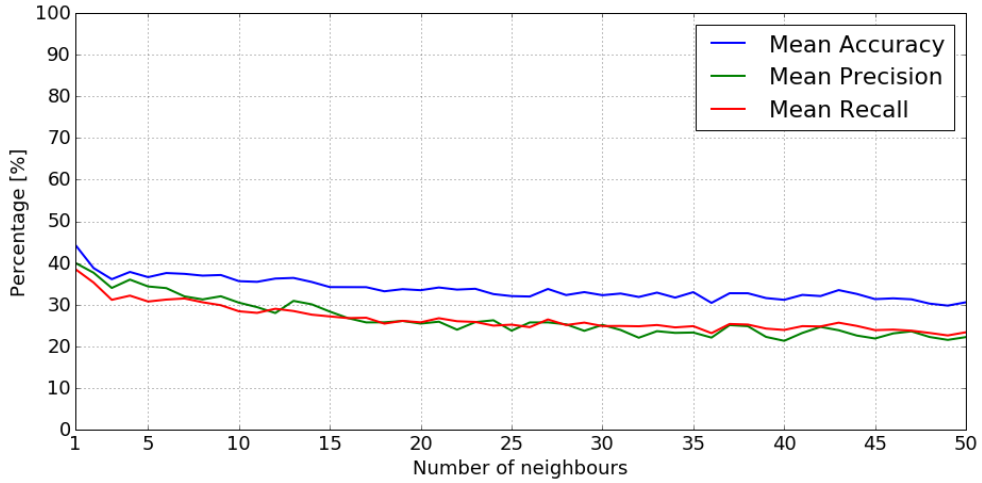
4.5.2 K-NN

K-NN ako jeden z najjednoduchších klasifikátorov som sa rozhodol testovať ako prvý. Testovaný bol na všetkých dátových súboroch obsahujúcich vyextrahované príznaky. Tento klasifikátor sa ukázal ako dobrý pre testovacie účely, no nie je až tak vhodný pre reálne podmienky, nakoľko klasifikácia vyžaduje výpočet vzdialeností medzi klasifikovanou vzorkou a všetkými natrénovanými vzorkami. Táto vlastnosť sa prejavila pri dátových súboroch obsahujúcich príznaky vyextrahované z fotografií z videa, nakoľko týchto fotografií bolo v základnom dátovom súbore 100 465. Testovanie bolo časovo náročné, keďže cez parameter K sme robili 25 meraní s hodnotami v rozmedzí 1 až 50, pričom každé meranie bolo vykonané 5 krát. Ako metrika vzdialeností jednotlivých vzoriek je použitá euklidovská vzdialenosť.

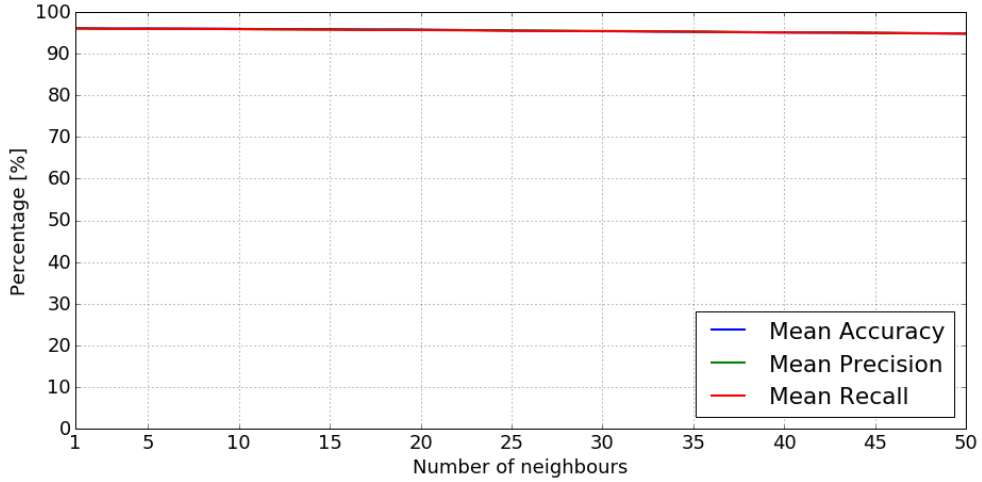


Obr. 4.10: Namerané metriky klasifikátoru *K-NN* pre rôzne hodnoty K na obrázkoch z *Google Street View*

Z obrázkov 4.10, 4.11 a 4.12 vidno, že klasifikátor najlepšie funguje na dátach z obrázkov z videa. Na týchto dátach klasifikátor dosahuje 95 % úspešnosť



Obr. 4.11: Namerané metriky klasifikátoru *K-NN* pre rôzne hodnoty K na obrázkoch z *Google Custom Search*



Obr. 4.12: Namerané metriky klasifikátoru *K-NN* pre rôzne hodnoty K na obrázkoch extrahovaných z videa

bez ohľadu na parameter K. Pre lepšie znázornenie výsledky taktiež uvádzam v tabuľke 4.5. Naopak najhoršie tento klasifikátor funguje na dátach z *Google Custom Search*, čo sa dá prisúdiť malému počtu dát. Ako najlepší parameter K sa javia hodnoty v rozmedzí od 2 do 5. Vyššie hodnoty znižujú všetky sledované metriky pod hranicu 80 %, čo pri použití *rejection rule* vedie k zamietnutiu aj správne klasifikovaných obrázkov.

Záverom tento klasifikátor pri použití dátového súboru vyextrahovaného zo snímkov z videa sa javí ako veľmi úspešný, no rýchlosť výpočtu a celkovo

K	Accuracy	Precision	Recall
1	95.89 %	96.09 %	95.92 %
11	95.67 %	95.83 %	95.70 %
21	95.42 %	95.57 %	95.43 %
31	95.15 %	95.27 %	95.17 %
41	94.88 %	94.99 %	94.90 %
50	94.64 %	94.77 %	94.66 %

Tabuľka 4.5: Namerané metriky klasifikátoru *K-NN* pre rôzne hodnoty K na obrázkoch extrahovaných z videa

validácie je pomalá. Najkratší čas, za ktorý sa vykonal jeden test bol 2 hodiny. Naopak najdlhší test trval 6 a pol hodiny, pričom počet záznamov a parametre testu (až na parameter K) boli konštantné.

4.5.3 Rozhodovacie Stromy

Rozhodovacie stromy sú ďalším z testovaných klasifikátorov. Konfigurované parametre modelu v tomto prípade sú: *výberové kritérium*, *rozdeľovacie kritérium* a maximálny počet príznakov, ktoré je možné použiť na rozdelenie uzlu. Tieto stromy nemajú obmedzenú hĺbku ani zadaný minimálny počet vzoriek na list. Z tohto dôvodu by mali predstavovať najlepšie naučený klasifikátor aký sa dá dosiahnuť pomocou základných rozhodovacích stromov.

Z výsledkov v tabuľkách 4.6, 4.7 a 4.8 je zrejmé, že mnou testované parametre klasifikátoru nijakým zásadným spôsobom neovplyvňujú úspešnosť modelu. Metriky tohoto stromu najviac ovplyvňuje maximálny počet príznakov použitých pri rozdelení, kde merania ktoré boli obmedzené log2 počtom príznakov, čo v tomto prípade znamená maximálne 9 príznakov, dosahujú najhoršie výsledky.

4.5.4 Ensemble Modely

Testované *ensemble modely* využívajú ako podmodel rozhodovacie stromy so základnými parametrami, kde ako výberové kritérium sa využíva *Gini*, rozdeľovacie kritérium je *best* a počet príznakov použitých k rozdeleniu uzlu nie je obmedzený. V predchádzajúcej časti som ukázal že tieto parametre nezohrávajú veľkú rolu pri trénovaní a že primárne záleží na použitom dátovom súbore. Pri testoch *ensemble modelov* testujem parameter určujúci počet trénovaných podmodelov. Tento počet je v rozmedzí od jedného do päťdesiatich podmodelov. Podobne ako pri *K-NN*, je počet meraní 25, pričom každý parameter je otestovaný päť krát.

[h]

criterion	splitter	max features	Accuracy	Precision	Recall
gini	**best**	**auto**	38.43 %	31.00 %	30.83 %
gini	**best**	**sqrt**	38.23 %	30.53 %	29.99 %
gini	**best**	**log2**	35.77 %	28.65 %	28.47 %
gini	**best**	**-**	41.69 %	33.86 %	33.12 %
gini	**random**	**auto**	35.29 %	27.87 %	27.92 %
gini	**random**	**sqrt**	35.50 %	28.75 %	28.59 %
gini	**random**	**log2**	32.38 %	25.84 %	25.40 %
gini	**random**	**-**	39.89 %	31.58 %	31.18 %
entropy	**best**	**auto**	38.91 %	32.11 %	31.64 %
entropy	**best**	**sqrt**	38.35 %	31.48 %	31.31 %
entropy	**best**	**log2**	36.05 %	29.91 %	29.45 %
entropy	**best**	**-**	42.16 %	35.64 %	35.10 %
entropy	**random**	**auto**	35.14 %	28.84 %	28.64 %
entropy	**random**	**sqrt**	34.71 %	28.28 %	27.79 %
entropy	**random**	**log2**	32.20 %	25.91 %	25.61 %
entropy	**random**	**-**	41.05 %	34.39 %	34.06 %

Tabuľka 4.6: Namerané metriky klasifikátoru *Rozhodovacie stromy* pre rôzne parametre na obrázkoch z *Google Street View*

4.5.4.1 Bagging

Prvým testovaným *ensemble modelom*, je teda *Bagging* s *Rozhodovacími stromami* ako podmodelom. Nakoľko tento test obsahuje viacero modelov, kde každý sa učí na rozdielnych množinách dát. Rozhodol tieto testy spustiť naraz, nakoľko sú časovo náročné ale medzi sebou nezávislé.

4.5.4.2 Boosting

Druhým testovaným *ensemble modelom* bol *Boosting*, opäť som ako podmodel využil *Rozhodovacie stromy*. Podobne ako pri *Bagging*-u som aj tieto testy vykonával paralelne, nakoľko nie sú závislé medzi sebou a z ich podstaty sú časovo náročné.

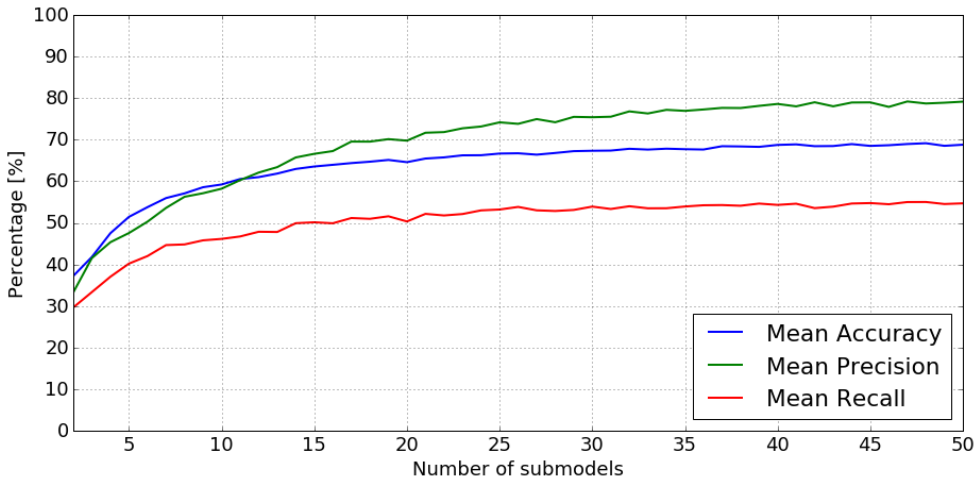
Z grafov 4.15, 4.16 a 4.17 môžeme vidieť, že tento typ klasifikátoru nie je vhodný, keďže ani jeden z testov neprekonal hranicu 50 %, pričom aj s rastúcim počtom podmodelov, klasifikátor stále vykazuje rovnaké metriky.

4.5.5 Viacvrstvové perceptrony

Ďalším z testovaných modelov boli viacvrstvové perceptronové siete (*MLP*). Tieto testy som implementoval čiastočne s použitím frameworku *TensorFlow* a čiastočne s využitím *scikit-learn*. Tento prístup som zvolil z dôvodu,

criterion	splitter	max features	Accuracy	Precision	Recall
gini	**best**	**auto**	26.27 %	22.53 %	21.85 %
gini	**best**	**sqrt**	27.43 %	23.81 %	23.07 %
gini	**best**	**log2**	25.58 %	22.36 %	21.91 %
gini	**best**	**-**	29.64 %	25.17 %	25.09 %
gini	**random**	**auto**	26.44 %	24.53 %	23.10 %
gini	**random**	**sqrt**	25.12 %	21.84 %	21.55 %
gini	**random**	**log2**	24.62 %	21.25 %	20.73 %
gini	**random**	**-**	28.75 %	24.37 %	23.89 %
entropy	**best**	**auto**	27.46 %	24.80 %	23.64 %
entropy	**best**	**sqrt**	25.25 %	22.47 %	20.85 %
entropy	**best**	**log2**	26.11 %	22.88 %	22.20 %
entropy	**best**	**-**	29.34 %	25.52 %	25.07 %
entropy	**random**	**auto**	25.15 %	22.44 %	21.42 %
entropy	**random**	**sqrt**	25.68 %	23.03 %	21.96 %
entropy	**random**	**log2**	24.52 %	21.73 %	21.00 %
entropy	**random**	**-**	26.50 %	23.58 %	22.45 %

Tabuľka 4.7: Namerané metriky klasifikátoru *Rozhodovacie stromy* pre rôzne parametre na obrázkoch z *Google Custom Search*



Obr. 4.13: Namerané metriky ensemble klasifikátoru *Bagging w/ Decision Tree* pre rôzne hodnoty K na obrázkoch z *Google Street View*

že *scikit-learn* obsahuje len základnú implementáciu *MLP*. Keďže som sa rozhodol testovať aj sieť s *DenseBlock* a veľkosť niektorých dátových súborov presahuje pamäť dostupných počítačov, rozhodol som sa siahnuť po frameworku *TensorFlow*, ktorý oba tieto problémy vyrieši.

criterion	splitter	max features	Accuracy	Precision	Recall
gini	**best**	**auto**	72.85 %	73.20 %	73.15 %
gini	**best**	**sqrt**	73.29 %	73.51 %	73.45 %
gini	**best**	**log2**	70.16 %	70.50 %	70.44 %
gini	**best**	**-**	77.52 %	77.66 %	77.50 %
gini	**random**	**auto**	70.67 %	71.00 %	71.04 %
gini	**random**	**sqrt**	70.73 %	70.94 %	71.00 %
gini	**random**	**log2**	67.16 %	67.45 %	67.54 %
gini	**random**	**-**	76.36 %	76.71 %	76.56 %
entropy	**best**	**auto**	74.68 %	74.83 %	74.94 %
entropy	**best**	**sqrt**	74.05 %	74.37 %	74.52 %
entropy	**best**	**log2**	71.82 %	72.12 %	72.25 %
entropy	**best**	**-**	78.06 %	78.18 %	78.24 %
entropy	**random**	**auto**	71.38 %	71.71 %	71.74 %
entropy	**random**	**sqrt**	71.37 %	71.66 %	71.80 %
entropy	**random**	**log2**	68.13 %	68.45 %	68.51 %
entropy	**random**	**-**	77.10 %	77.32 %	77.35 %

Tabuľka 4.8: Namerané metriky klasifikátoru *Rozhodovacie stromy* pre rôzne parametre na obrázkoch extrahovaných z videa

4.5.5.1 Štandardné MLP

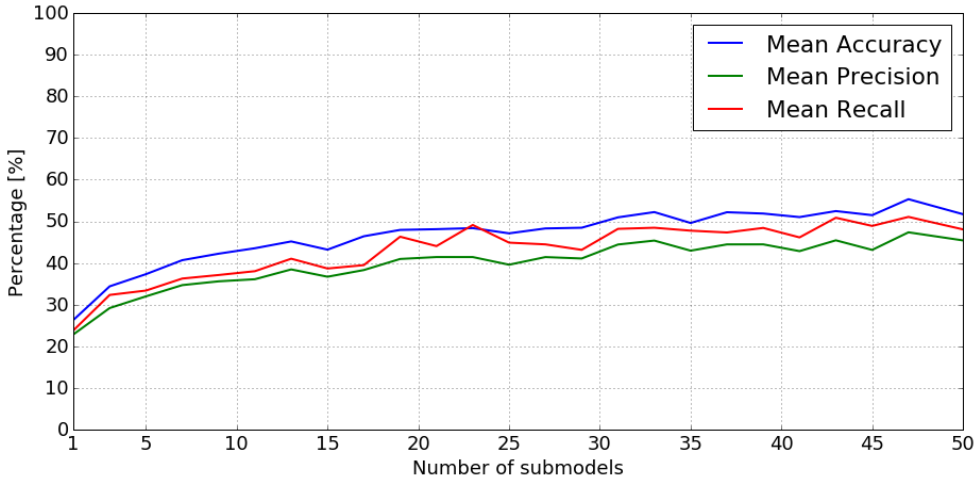
Testované parametre štandardného *MLP* boli aktivačná funkcia, veľkosť skrytej vrstvy a optimalizátor. Testované aktivačné funkcie boli *ReLU*, *Identita*, *Sigmoida* sieť bez aktivačnej funkcie, ako optimalizátory som zvolil štandardné *SGD* a *Adam*. Posledným parametrom je veľkosť skrytej vrstvy, hodnota tohto parametru je z rozmedzia 100 až 1 500. Nakoľko sa ukázalo, že vyššie hodnoty nemajú takú úspešnosť. Počet trénovacích cyklov bol nastavený na 2 000. Pokiaľ sa počas dvoch iterácií nezlepšila hodnota chybovej funkcie (*Cost*) o viac ako 10^{-4} bolo trénovanie predčasne ukončené. Toto ošetrenie má predchádzať uviaznutiu v lokálnom minime.

Vzhľadom na počet testov však uvediem len výsledky pre optimalizátor *Adam* a aktivačné funkcie *ReLU* a *Sigmoidu*. Výsledky ostatných testov je možné nájsť na médiu k tejto práci.

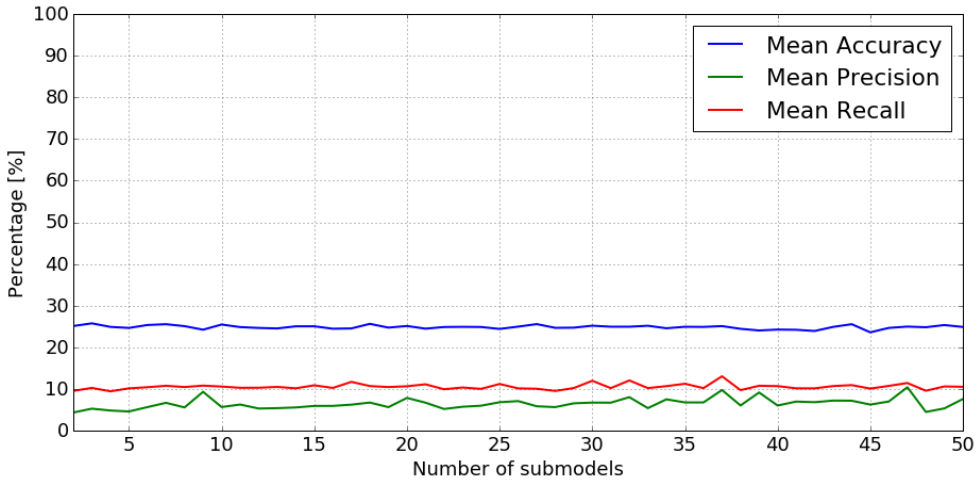
4.5.5.2 MLP s *DenseBlock*

Druhou testovanou sieťou bol *MLP* s *DenseBlock*. Takáto sieť prináša do pôvodného konceptu viacvrstvovej perceptronovej siete ďalšie nelineárne prvky mimo aktivačných funkcií.

Pri testovaní tohto druhu siete som ako premenlivé parametre zvolil veľkosť skrytej vrstvy a veľkosť bloku. Veľkosť bloku v tomto prípade znamená



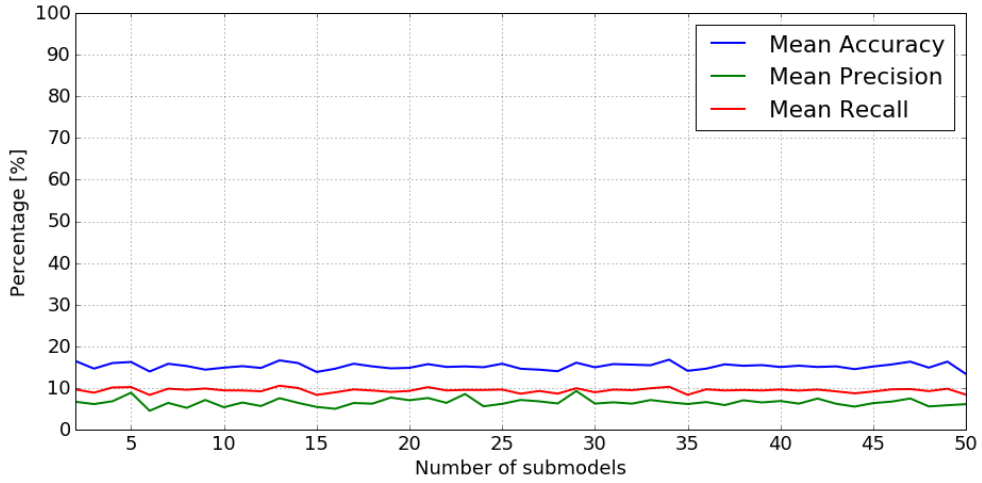
Obr. 4.14: Namerané metriky ensemble klasifikátoru *Bagging w/ Decision Tree* pre rôzne hodnoty K na obrázkoch z *Google Custom Search*



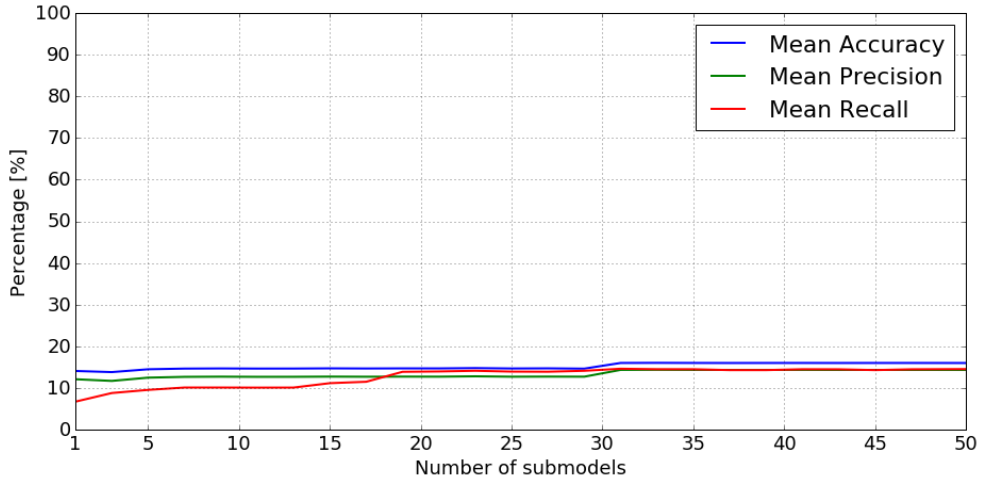
Obr. 4.15: Namerané metriky ensemble klasifikátoru *Boosting w/ Decision Tree* pre rôzne hodnoty K na obrázkoch z *Google Street View*

počet násobiacich vrstiev (plne prepojených vrstiev), ktoré sú za sebou, pričom za každou sa k jej výsledku pripojí výsledok z predchádzajúcich násobiacich vrstiev.

Parameter určujúci veľkosť skrytej vrstvy som teda volil postupne 100, 500, 1 000, 1 500 a 2 000. Vyššia hodnota sa ukázala za zbytočnú nakoľko už pri 500 testovaný model vykazoval metriku *Accuracy* vyššiu než 90 %. Druhým parametrom, bol počet prepojených vrstiev. Hodnotu tohto parametra som volil postupne 1, 5 a 10. Opäť pri testoch sa preukázalo, že voliť vyššiu hodnotu



Obr. 4.16: Namerané metriky ensemble klasifikátoru *Boosting w/ Decision Tree* pre rôzne hodnoty K na obrázkoch z *Google Custom Search*

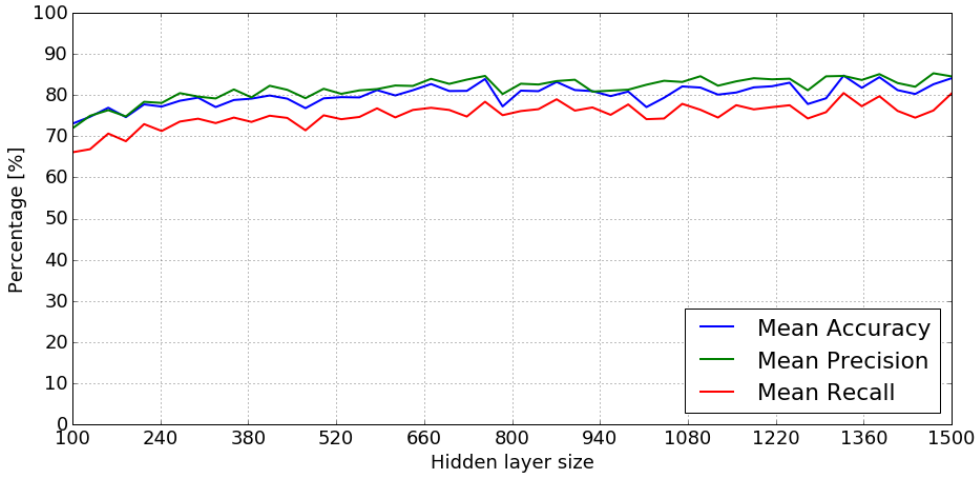


Obr. 4.17: Namerané metriky ensemble klasifikátoru *Boosting w/ Decision Tree* pre rôzne hodnoty K na obrázkoch extrahovaných z videa

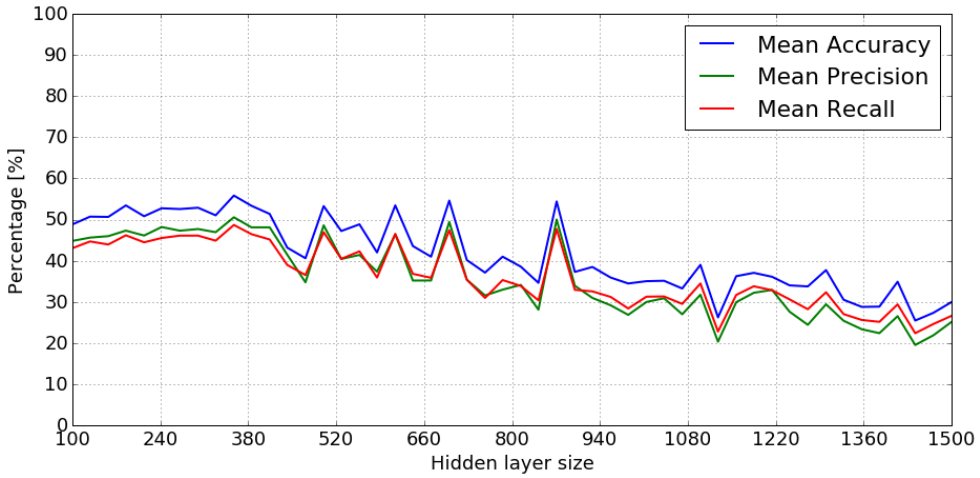
je zbytočné, nakoľko model nebol schopný sa naučiť na viac ako 90 % pri počte prepojených vrstiev rovnom 10.

Ako aktivačné funkcie skrytých vrstiev som použil *ReLU*. Táto funkcia sa ukázala za vhodnú, nakoľko aj pri testoch so štandardnými viacvrstvovými perceptronami vykazovala vysoké metriky.

Namerané výsledky je možné zhliadnuť v tabuľke 4.9, v rámci ďalších meraní som experimentoval s aktivačnými funkciami *Tanh* a *Sigmoida*, výsledky týchto meraní sa nachádzajú na priloženom médiu.



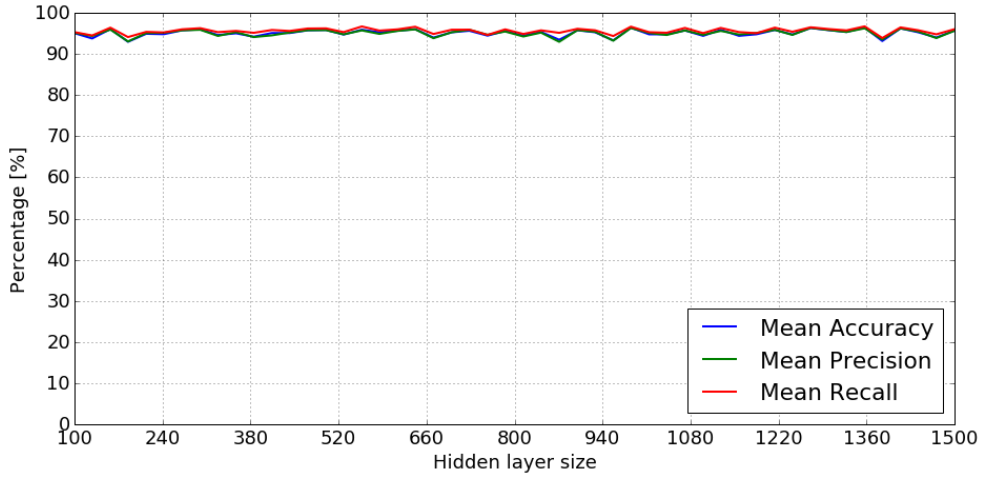
Obr. 4.18: Namerané metriky klasifikátoru *MLP* s aktivačnou funkciou *ReLU* a optimalizátorom *Adam* pre rôzne veľkosti skrytej vrstvy na obrázkoch extrahovaných z *Google Street View*



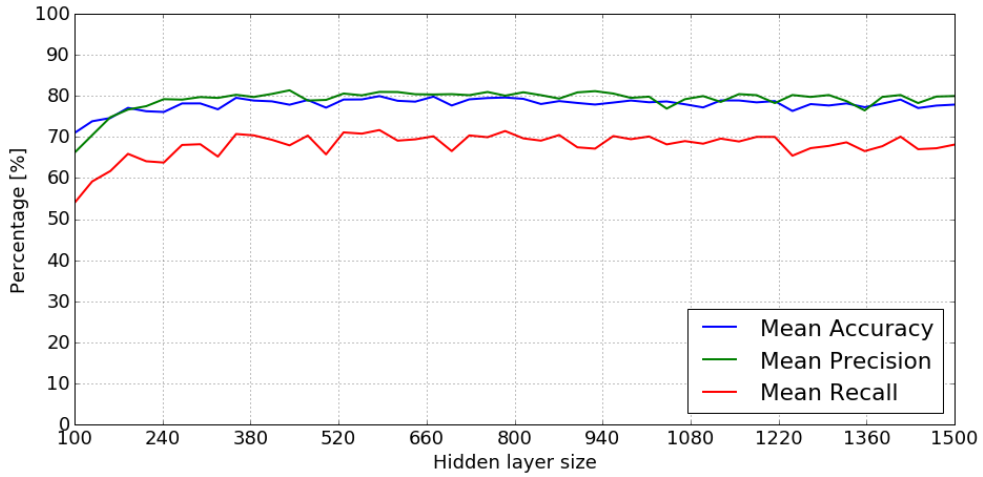
Obr. 4.19: Namerané metriky klasifikátoru *MLP* s aktivačnou funkciou *ReLU* a optimalizátorom *Adam* pre rôzne veľkosti skrytej vrstvy na obrázkoch extrahovaných z *Google Custom Search*

Tento klasifikátor bol však testovaný len na dátach z *Google Street View*, nakoľko sa jednalo o testovanie konceptu a jeho porovnanie so štandardným *MLP*. Keďže výsledky sa nepreukázali razantne lepšie (rozdiel medzi modelmi je približne 5 %), rozhodol som sa v testovaní nepokračovať na dátových súboroch s dátami extrahovanými zo snímkov z videa.

Z nameraných hodnôt v tabuľke 4.9, som usúdil že tento klasifikátor je

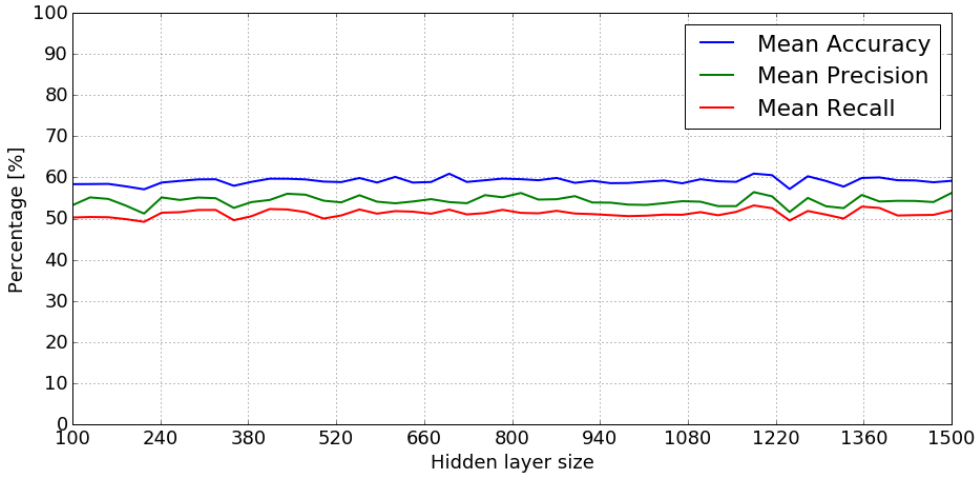


Obr. 4.20: Namerané metriky klasifikátoru *MLP* s aktivačnou funkciou *ReLU* a optimalizátorom *Adam* pre rôzne veľkosti skrytej vrstvy na obrázkoch extrahovaných z videa

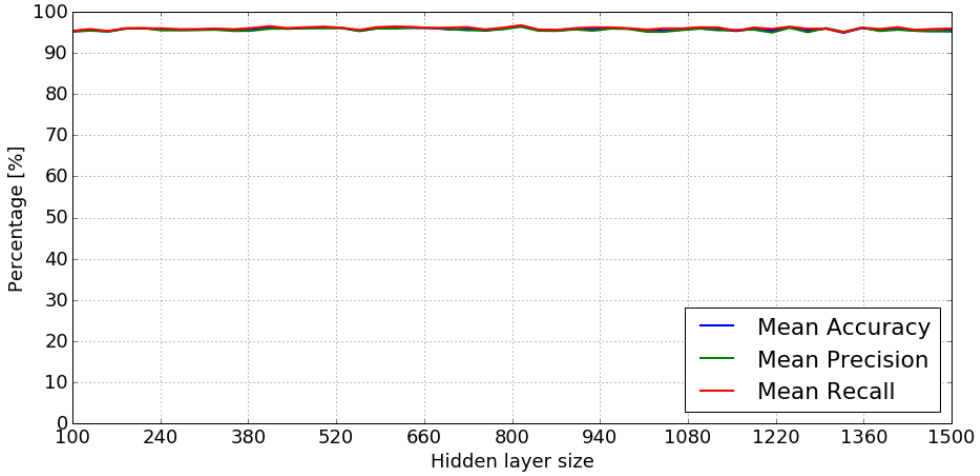


Obr. 4.21: Namerané metriky klasifikátoru *MLP* s aktivačnou funkciou *Sigmoid* a optimalizátorom *Adam* pre rôzne veľkosti skrytej vrstvy na obrázkoch extrahovaných z *Google Street View*

síce schopný naučiť sa na tomto dátovom súbore, ale primárne parametere, ktorými sú veľkosť skrytej vrstvy a počet *DenseBlock*, nijak zásadne neovplyvňujú metriky klasifikácie. Dalo by sa povedať, že rozdiel medzi nameranými hodnotami pre fixnú veľkosť skrytej vrstvy a rôzne hodnoty počtu prepojených vrstiev je štandardná odchylka vyplývajúca zo spôsobu učenia modelu.



Obr. 4.22: Namerané metriky klasifikátoru *MLP* s aktivačnou funkciou *Sigmoid* a optimalizátorom *Adam* pre rôzne veľkosti skrytej vrstvy na obrázkoch extrahovaných z *Google Custom Search*



Obr. 4.23: Namerané metriky klasifikátoru *MLP* s aktivačnou funkciou *Sigmoid* a optimalizátorom *Adam* pre rôzne veľkosti skrytej vrstvy na obrázkoch extrahovaných z videa

4.5.6 Hlboké konvolučné siete

Posledným zo základných testov boli testy hlbokých konvolučných sietí. Tieto testy prebiehali na obrázkoch z *Google Street View* a na obrázkoch zo snímkov z videa. Testy bežali 24 hodín, výpočty boli vykonávané na strojoch s grafickou kartou s podporou *CUDA*. Väčšina testovaných modelov však nebola schopná naučiť sa dodaných dátach lepšie než na 20 % presnosti (*Accuracy*).

Hidden layer size	Dense Block size	Activation	Accuracy	Precision	Recall
100	**1**	**relu**	85.69 %	83.00 %	81.60 %
100	**5**	**relu**	84.01 %	82.04 %	79.79 %
100	**10**	**relu**	79.51 %	78.06 %	74.08 %
500	**1**	**relu**	90.24 %	90.14 %	85.78 %
500	**5**	**relu**	90.46 %	90.12 %	87.64 %
500	**10**	**relu**	89.75 %	89.54 %	85.63 %
1 000	**1**	**relu**	89.84 %	90.56 %	86.73 %
1 000	**5**	**relu**	91.27 %	90.26 %	87.94 %
1 000	**10**	**relu**	90.27 %	89.99 %	86.66 %
1 500	**1**	**relu**	91.29 %	90.90 %	87.78 %
1 500	**5**	**relu**	91.41 %	91.25 %	89.30 %
1 500	**10**	**relu**	91.04 %	90.07 %	89.01 %
2 000	**1**	**relu**	91.08 %	90.34 %	88.17 %
2 000	**5**	**relu**	89.79 %	88.77 %	86.04 %
2 000	**10**	**relu**	90.37 %	90.04 %	88.88 %

Tabuľka 4.9: Namerané metriky klasifikátoru *MLP* s *DenseBlock* pre rôzne parametre na obrázkoch extrahovaných z *Google Street View*

Štruktúru použitých sietí je možné si prehliadnuť v prílohe C.

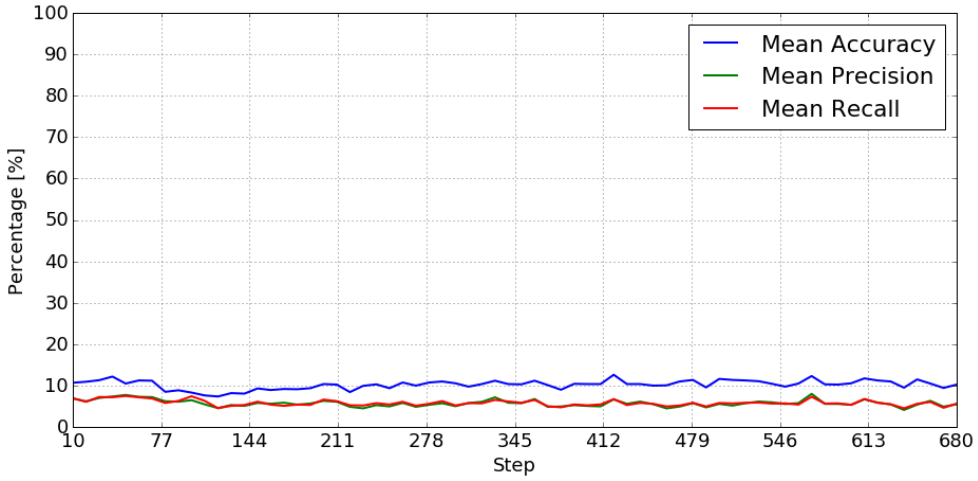
4.5.6.1 Alexnet

Prvou testovanou sieťou bol *AlexNet*, trénovaná bola na dvoch dátových súboroch. Prvým boli obrázky z *Google Street View*. Druhým obrázky zo snímkov z videa. Vzhľadom na počet obrázkov zo snímkov z videa (100 465), tento test vykonal celkovo 90 iterácií za 24 hodín. Trénovanie na obrázkoch z *Google Street View* bolo o niečo málo úspešnejšie. Za 24 hodín vykonalo 680 iterácií a úspešnosť klasifikácie bola 10 % oproti 8 % pri obrázkoch zo snímkov z videa.

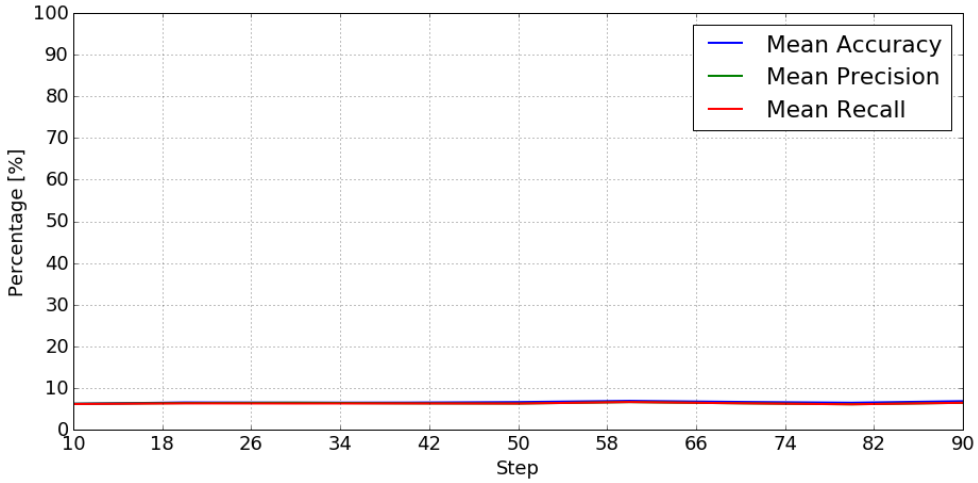
Vzhľadom na namerané výsledky a čas potrebný k trénovaniu som sa rozhodol tento typ siete ďalej neskúmať, nakoľko po 24 hodinách sa merané metriky zmenili o minimálnu hodnotu.

4.5.6.2 Dvojitý Alexnet

Ďalším testovaným typom siete bol *Dvojitý AlexNet*. Táto sieť je podobná *AlexNet*-u s tým rozdielom že za poslednú konvolučnú vrstvu v pôvodnej architektúre je pridaný ešte jeden blok konvolučných vrstiev opäť zakončený *Max Pooling* vrstvou. Architektúru tejto siete je možné pozrieť si na obrázku C.1 v prílohe C.



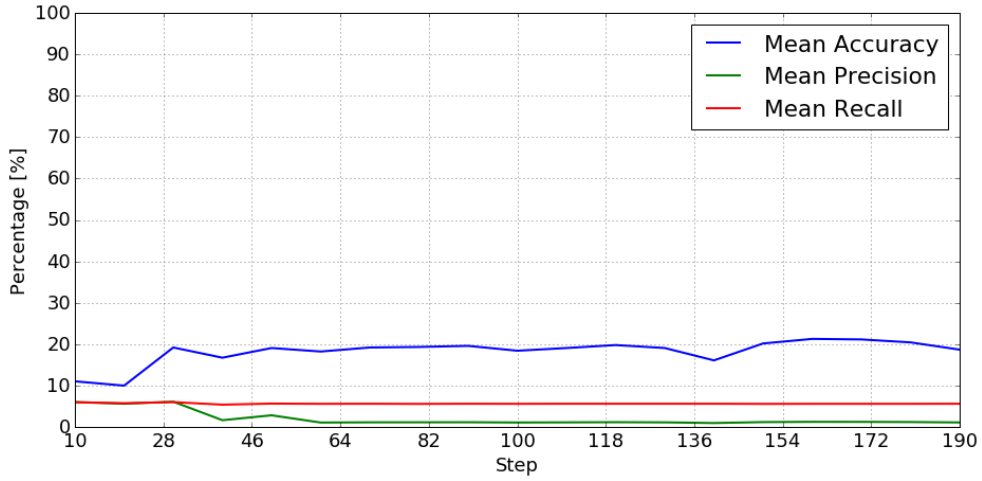
Obr. 4.24: Namerané metriky klasifikátoru *Alexnet* na obrázkoch extrahovaných z *Google Street View*



Obr. 4.25: Namerané metriky klasifikátoru *AlexNet* na obrázkoch extrahovaných z videa

Vzhľadom na predchádzajúce meranie siete *AlexNet* som sa rozhodol tento test vykonať len na dátach z *Google Street View* a v prípade vyššej úspešnosti tento model následne otestovať aj na snímkoch z videa.

Podobne ako pri obyčajnej sieti *AlexNet*, aj sieť typu dvojitý *AlexNet* nebola schopná naučiť sa na obrázkoch z *Google Street View*, z tohoto dôvodu som ďalej v testovaní tejto siete nepokračoval. V porovnaní s *AlexNet* táto sieť dosiahla lepšieho výsledku, ale stále nedostatočného.



Obr. 4.26: Namerané metriky klasifikátoru *Dvojitý AlexNet* na obrázkoch extrahovaných z *Google Street View*

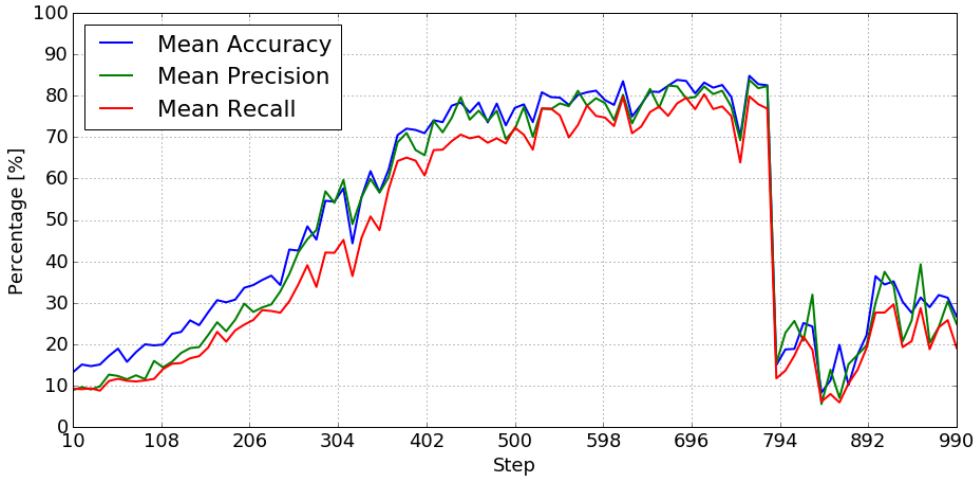
4.5.6.3 Vlastný návrh siete

Posledným testovaným typom hlbokej konvolučnej siete je môj vlastný návrh, opäť je možné pozrieť si jej návrh na obrázku C.1 v prílohe C. Rozdiel medzi dvojitým *AlexNet*-om a mojou architektúrou spočíva v pridanej ďalšej konvolučnej vrstve a v rozdielnych parametroch jednotlivých konvolučných vrstiev.

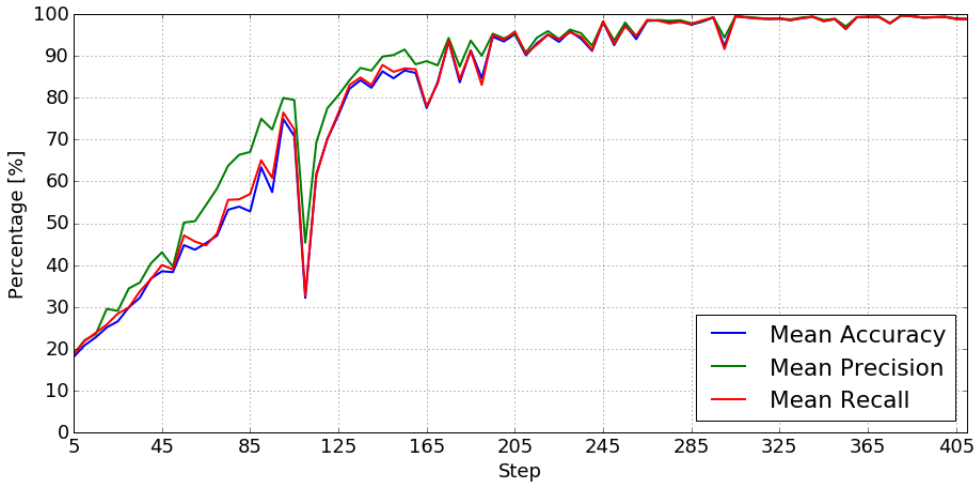
Tento typ siete som pomenoval *MyNetwork*, pre jednoduché označenie v texte. Pri učení tejto siete na snímkoch z videa som upravil trénovací cyklus, pretože súbor týchto obrázkov je veľký. Použitie všetkých fotografií na jeden trénovací cyklus mi prišlo ako nevhodný prístup, rozhodol som sa nastaviť konštantný počet *Batch*-ov miesto vypočítaného. *Batch size* som zvolil 100 a pretože celkový počet obrázkov v tomto dátovom súbore je 100 465, vychádza celkový počet *Batch*-ov na 1 004, táto hodnota je však príliš vysoká na trénovanie konvolučných sietí, už len z dôvodu dĺžky jedného trénovacieho cyklu, rozhodol som sa použit 100 *Batch*-ov, na jeden trénovací cyklus miesto spomínaných 1 004.

Pre ukážku uvádzam jednotlivé aktivácie na vstupnom obrázku v kapitole D, aktivácie pochádzajú zo siete pretrénovanej na snímoch z mobilu, priebeh učenia sa nachádza na grafe 4.28.

Z výsledkov zobrazených na grafoch 4.27 a 4.28 vidno, že táto sieť je schopná sa naučiť rozpoznávať vybrané pamiatky. Trénovanie oboch sietí som zastavil po dvanástich hodinách. Na konci grafu na obrázku 4.27 vidíme rapídny pokles metrík, táto sieť ďalej začala stagnovať a uviazla v lokálnom optime, ktorého metriky boli pod 20 %. Vzhľadom na úspešnosť pri testovaní som sa rozhodol túto sieť otestovať aj na snímkoch z videa, čo preukázalo vhodný návrh tejto siete.



Obr. 4.27: Namerané metriky klasifikátoru *MyNetwork* na obrázkoch extrahovaných z *Google Street View*



Obr. 4.28: Namerané metriky klasifikátoru *MyNetwork* na obrázkoch extrahovaných z videa

4.6 Výber modelu

Predposlednou častou realizácie bol výber modelu a prípadne typu predtrénovanej siete. Z testovaných modelov najlepšie vyšli štandardný viacvrstvový perceptron s jednou skrytou vrstvou a môj návrh hlbokej konvolučnej siete. Tieto modely som sa rozhodol odtestovať pomocou krížovej validácie aj na derivátoch fotografií zo snímkov z videa. Tieto upravené fotografie obsahujú rôzne rotácie, zmeny jasu, zaostrenie a rozostrenie obrázku. Celkovo obsahujú

okolo 2 000 000 fotografií. Testy boli opäť vykonané s použitím *StratifiedKFold* algoritmu s parametrom K rovným 5.

Dáta pre obe krížové validácie boli najprv predpripravené nakoľko trénovanie oboch sietí, pri taktomto vysokom počte záznamov je časovo velmi náročné. Výsledné trénovacie a testovacie súbory som vytvoril tak aby z každého súboru fotografií, ako pôvodných fotografií tak aj derivátov, bolo vo výsledých súboroch rovnaké zastúpenie.

Samotná krížová validácia tak prebiehala paralelne, na všetkých rozdeleniach súčasne.

4.6.1 Krížová validácia *MyNetwork*

Krížová validácia hlbokej konvolučnej siete *MyNetwork*, trvala 18 hodín na jednu sieť. Ani jezda zo sietí však nebola schopná sa naučit rozpoznávať zvolené pamiatky ani po 400 trénovacích cykloch. Priemerný výsledok validácie bol 8 %. Celé namerané výsledky sa nachádzajú tabulke 4.6.1. Priebeh učenia prvého preloženia je možné vidiet na obrázku4.29. Ostatné priebehy sa nachádzajú v prílohe F.

Fold Number	**Accuracy**	**Precision**	**Recall**
1	6.830 %	0.426 %	6.25 %
2	6.832 %	0.427 %	6.25 %
3	8.869 %	0.554 %	6.25 %
4	8.869 %	0.525 %	6.25 %
5	8.850 %	0.510 %	6.25 %
Mean	8.05 %	0.488 %	6.25 %

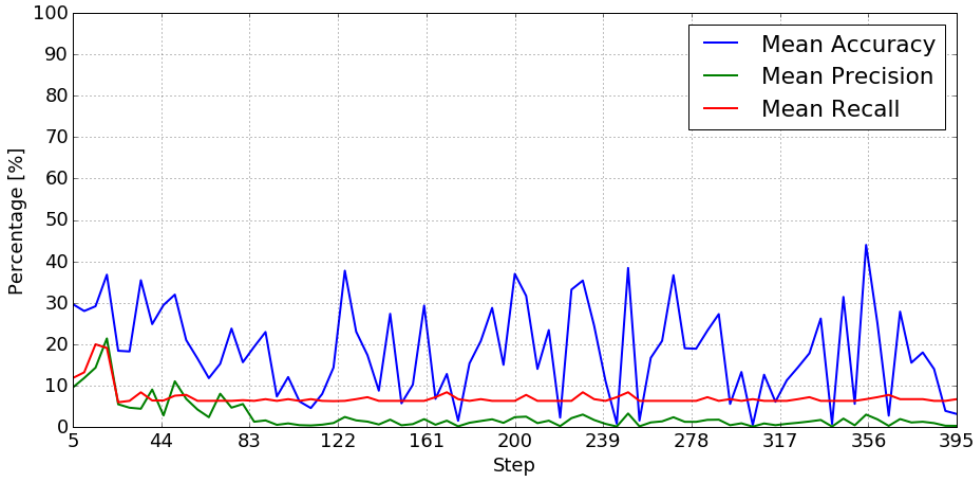
Tabuľka 4.10: Výsledky krížovej validácie na klasifikátore *MyNetwork* s použitím celého súboru vygenerovaných obrázkov zo snímkov z videa

4.6.2 Krížová validácia *MLP*

Krížová validácia *MLP* prebiehala s využitím frameworku *scikit-learn* a viacprocserového prístupu. Opäť ako pri krížovej validácií *MyNetwork*, som si predpripravil vstupné dátove súbory a jednotlivé testy preložení spustil paralelne. Tieto testy využívajú ako vstupný dátový súbor príznaky extrahované zo snímkov z videa a deriváty týchto snímkov. Jeden trénovací dátový súbor má 1 526 956 a testovací 381 879. Výsledky merania sa nachádzajú v tabuľke 4.6.2.

4.6.3 Výber trénovacieho súboru dát

Vzhľadom na to, že krížová validácia ukázala, že prístup pomocou prvotnej extrakcie príznakov pomocou hlbokých konvolučných sietí je vhodnejší,



Obr. 4.29: Priebeh učenia krížovej validácie klasifikátoru *MyNetwork* pri prvom preložení

Fold Number	Accuracy	Precision	Recall
1	72.31 %	68.82 %	67.91 %
2	75.12 %	70.40 %	68.65 %
3	71.47 %	68.52 %	69.31%
4	74.82 %	69.81 %	68.36 %
5	70.79 %	67.32 %	65.24 %
Mean	72.90 %	68.974 %	67.894 %

Tabuľka 4.11: Výsledky krížovej validácie na klasifikátore *MLP* s použitím celého súboru vygenerovaných obrázkov zo snímkov z videa

rozhodol som sa otestovať rôzne spomínané siete. Tieto siete sú *DenseNet-121*, *DenseNet-161*, *DenseNet-169*, *DenseNet-201*, *ResNet-50*, *ResNet-101* a *Resnet-152*. Tieto merania som vykonal na obrázkoch z *Google Street View*, opäť pomocou krížovej validácie *StartifiedKFold* s piatimi preloženiami. Testy využivajú *MLP* ako klasifikátor a opäť iterujú cez parameter veľkosti skrytej vrstvy. Ostatné parametre sú fixne nastavené na hodnoty: optimalizátor *Adam* a aktivačná funkcia *ReLU*. Spriemerované namerané výsledky sa nachádzajú v tabuľke 4.6.3. Grafy jednotlivých meraní je možné si pozriet v prílohe G.

Ako najlepšia z týchto sietí vyšla *ResNet-50*, pričom rozdiel medzi sietami typu *ResNet* je možné považovať za chybu modelu.

4.6.4 Konečný výber modelu

Vzhľadom na výsledky meraní v predchádzajúcich častiach som sa rozhodol zvoliť ako model viacvrstvový perceptron s jednou skrytou vrstvou, tento mo-

Dataset	Accuracy	Precision	Recall
DenseNet-121	80.06 %	81.69 %	75.20 %
DenseNet-161	90.87 %	91.29 %	88.50 %
DenseNet-169	84.64 %	85.38 %	80.82 %
DenseNet-201	88.17 %	88.35 %	84.96 %
ResNet-50	97.83 %	97.69 %	96.43 %
ResNet-101	97.72 %	97.41 %	96.21 %
ResNet-152	97.76 %	97.47 %	96.36 %

Tabuľka 4.12: Výsledky meraní klasifikátoru *MLP* na obrázkoch z *Google Street View* s extrakciou príznakov pomocou rôznych hlbokých konvolučných sietí

del sa ukázal ako vhodný kandidát s jednoduchou možnosťou úpravy jeho parametrov a štruktúry samotnej siete.

Ako model hlbokej konvolučnej siete na predspracovanie som k nemu zvolil *Resnet-50* nakoľko vyššiel z testovaných hlbokých konvolučných sietí najlepšie.

Pre otestovanie mojej voľby som klasifikátor *MLP* naučil na dátovom súbor obsahujúcom príznaky zo snímkov z videa a následne som meral metriky klasifikácie dátových súborov obsahujúcich príznaky derivátov fotografií. Výsledky týchto meraní sa nachádzajú v tabuľke 4.6.4.

4.7 Integrácia

Poslednou časťou realizácie bola integrácia navrhnutého modelu s vybranými parametrami do chatbota vyvíjaného u nás na fakulte. Môj základný návrh obsahoval rozdelenie integrácie na dve časti. Prvou je vytvorenie obecného "frameworku"ktorý má za úlohu načítať model a poskytnúť jednoduché API pre klasifikáciu dodaného obrázku. Naviac pri štarte by mal vykonať krížovú validáciu a poskytovať metriky tejto validácie. Druhá časť je napojenie tohoto frameworku na chatbota *Golem*.

4.7.1 Framework

Samotný framework je riešený ako klient/server aplikácia, ktorá naslúcha na *unix socket*, na ktorý prichádzajú jednotlivé požiadavky. Požiadavok je určený príkazom a prípadne dodaným argumentom. Tieto príkazy sú jednoduché, ako napríklad príkaz *PREDICT*, ktorý slúži, ako už z názvu vyplýva, na klasifikáciu argumentu, ktorý je potrebný. Aplikácia na obsluhu klienta využíva separátne vlákno, čiže je možnosť obsluhy viacerých klientov naraz, čo je v tomto prípade žiadané, nakoľko po napojení na *Chatbota* sú jednotlivé požiadavky asynchrónne a teda môže ich prísť viac naraz, prípadne počas obsluhy nejakého iného požiadavku.

Dataset	Accuracy	Precision	Recall
Rotated by -22.0 °	85.16 %	90.27 %	85.16 %
Rotated by -13.2 °	91.72 %	93.28 %	91.72 %
Rotated by -4.4 °	94.39 %	95.16 %	94.39 %
Rotated by 4.4 °	94.67 %	95.33 %	94.67 %
Rotated by 13.2 °	91.34 %	93.85 %	91.34 %
Rotated by 22.0 °	82.91 %	91.01 %	82.91 %
Brightness -100	73.63 %	82.29 %	73.63 %
Brightness -60	88.62 %	90.65 %	88.62 %
Brightness -20	95.09 %	95.57 %	95.09 %
Brightness +20	93.50 %	94.50 %	93.50 %
Brightness +60	85.43 %	89.93 %	85.43 %
Brightness +100	64.22 %	79.50 %	64.22 %
Sharpen by 8.0	70.18 %	79.07 %	70.18 %
Sharpen by -6.5	75.18 %	82.20 %	75.18 %
Sharpen by -5	80.87 %	85.53 %	80.87 %
Blur by 3	96.04 %	96.38 %	96.04 %
Blur by 9	97.55 %	97.70 %	97.55 %
Blur by 17	97.84 %	97.99 %	97.84 %

Tabuľka 4.13: Výsledky meraní klasifikátoru *MLP* naučenom na obrázkoch zo snímkov z videa a meranom na rôznych derivátoch týchto fotografií

4.7.1.1 Základný popis inicializácie

Základný algoritmus aplikácie je jednoduchý a priamočiary. Ako prvé aplikácia načíta trénovací *CSV* súbor a rozdelí ho na pole príznakov a k nim prislúchajúcich tried.

Z týchto dát sa následne vypočítajú informácie o vstupom dátovom súbore, ako je počet vzoriek, počet atribútov, počet tried a počty vzoriek prislúchajúcich jednotlivým triedam. Tieto informácie sú neskôr poskytované pomocou príkazu *DATASET INFO*, ktorý si popíšeme v ďalšej kapitole.

Následne sa vytvorí trieda reprezentujúca klasifikačný postup, táto trieda sa stará o predspracovanie fotografie, extrakciu príznakov a následnú klasifikáciu, jedná sa teda o triedu zlučujúcu všetky časti klasifikačného procesu do jedného. Trieda je písaná tak aby bola modularizovateľná a aby sa jednotlivé časti klasifikačného procesu dali čo najjednoduchšie zameniť za iné.

Po vytvorení tejto triedy nasleduje spustenie krížovej validácie na trénovacom súbore, táto prebieha pomocou *Stratified K-Fold* algoritmu s piatimi preloženiami. Počas tejto validácie sa ukladajú najlepšie namerané metriky a rozdelenie trénovacej a testovacej množiny, pri ktorej boli namerané. Výsledkom tohoto procesu sú metriky ktoré sú opäť neskôr dostupné pomocou jednoduchého príkazu *CROSS VALIDATION RESULTS*.

Po dokončení krížovej validácie nasleduje samotné trénovanie modelu na trénovacích dátach. Toto je závislé od použitého modelu, či už sa jedná o *K-NN*, *MLP* alebo sieť vytvorenú pomocou frameworku *TensorFlow*. V tejto fázy je možné miesto trénovania načítať uložený model.

Ďalším krokom je validácia doplnkových dátových súborov. Keďže v testovacej fázy som si predpripravil viacero testovacích dátových súborov obsahujúcich rôzne variácie pôvodných obrázkov, ktoré boli modifikované, či už zmenou jasu, rotáciou alebo rozmazaním. Rozhodol som sa tieto testovacie dátové súbory taktiež zahrnúť do validácie pre kontrolu kvality modelu. Táto validácia prebieha na modely naučenom v predchádzajúcom kroku, pričom dáta sú taktiež dostupné pomocou príkazu *VALIDATED DATASET RESULTS*.

Posledným krokom je samotné vytvorenie *socketu* a spustenie serveru. Tento krok využíva štandardných systémových knižníc jazyka *Python*, pričom je navrhnutý tak aby umožňoval asynchrónnu obsluhu viacerých požiadaviek.

4.7.1.2 Obsluha klienta

Akonáhle sa klient pripojí na môj server, okamžite je preňho vytvorené obslužné vlákno. Vlákno ako prvé inicializuje potrebné dátové štruktúry, ktoré zväčša obsahujú statické dáta predvypočítané, ktoré by mohol klient požadovať. Po inicializácií sa načítavajú postupne jednotlivé príkazy. Tieto príkazy sú v obyčajnej textovej forme. Každý príkaz je ukončený novým riadkom. Argumenty pre jednotlivé príkazy sú taktiež ukončované novými riadkami a nachádzajú sa hneď za príkazom. Po prečítaní príkazu sa najprv skontroluje či je príkaz valídny, pokiaľ nie, je klientovy odoslaná chybová hláška a spojenie je ukončené. Ak je zadaný príkaz valídny, nastupuje jeho spracovanie, počas ktorého sa načítajú potrebné argumenty a príkaz sa vykoná. Obecne na každý príkaz je odoslaná odpoveď vo formáte *JSON*, základ tejto správy tvorí status, ktorý určuje, či bol daný príkaz vykonaný úspešne. Pokiaľ nie, tak obsahuje správu, ktorá popisuje vzniknutú chybu. Klient môže zadať viacero príkazov, no všetky musia byť valídne, inak nastane ukončenie spojenia.

4.7.1.3 Príkazy

Implementovými príkazmi sú *PREDICT*, *PREDICT MULTIPLE*, ďalšie príkazy slúžia k popísaniu aktuálnej konfigurácie a metrík validácie. Sú to príkazy *DATASET INFO*, *CROSS VALIDATION RESULTS*, *VALIDATED DATASET RESULTS*. Ďalej sú tu pomocné príkazy na vypnutie aplikácie, otestovanie spojenia a iné. Návratové hodnoty týchto príkazov sú *JSON* objekty, nakoľko dáta, ktoré aplikácia vracia sú zväčša polia, prípadne dvojice hodnôt. Výhodou tejto reprezentácie je taktiež jej rozšírená implementácia v rôznych jazykoch, čím je zabezpečená prenositeľnosť medzi rôznymi jazykmi.

Príkazy *PREDICT* a *PREDICT MULTIPLE*
Ako už z názvu vyplýva slúžia na klasifikáciu obrázkov dodaných ako argument. Obrázky môžu byť v dvoch variáciách, buď ako odkaz na obrázok na internete alebo ako obrázok zakódovaný v *Base64*. Návratovou hodnotou týchto príkazov je chybová hláška alebo výsledok klasifikácie popisujúci, či bol daný obrázok úspešne zaradený alebo zamietnutý, triedu, do ktorej bol obrázok zaradený a pravdepodobnosti zaradenia do jednotlivých tried.

Príkaz *DATASET INFO*
Tento príkaz slúži na vrátenie informácie o vstupnom súbore príznakov ako počet záznamov, počet príznakov, počet tried a rozdelenie záznamov do tried. Tieto dáta sú vypočítané dopredu kvôli urýchleniu obsluhy príkazu.

Príkaz *CROSS VALIDATION RESULUTS*
Návratovou hodnotou tohoto príkazu je výsledok krížovej validácie. Toto zahŕňa spriemerované metriky, ako *Accuracy*, *Recall* a *Precision* a metriky najlepšieho výsledku. Tieto metriky obsahujú opäť *Accuracy*, *Recall*, *Precision*, ďalej *Confusion Matrix*, *Precision* a *Recall* pre jednotlivé triedy a rozdelenie vstupných súborov príznakov na trénovaciu a testovaciu množinu. Tieto dáta sú zakódované opäť vo formáte *JSON* pre jednoduché spracovanie. Dáta sú opäť predpočítané pri spustení aplikácie, pre urýchlenie obsluhy.

Príkaz *VALIDATED DATASET RESULTS*
Nakoľko som počas práce vygeneroval niekoľko vstupných dátových súborov, rozhodol som sa implementovať do frameworku príkaz, ktorý vráti metriky validácie týchto dát. Validácia prebieha na modely natrénovanom na trénovacom vstupom súbore. Metriky týchto validácií sú však vypočítané na základe dátových súborov ktoré sa nepoužili na trénovanie modelu. Dátové súbory sú taktiež validované pri spustení aplikácie, nakoľko samotná validácia trvá rádovo v minútach. Vrátené hodnoty sú podobné ako pri predchádzajúcom príkaze a to *Accuracy*, *Precision*, *Recall* pre celý dátový súbor, *Confusion Matrix*, *Precision* a *Recall* pre jednotlivé triedy a nakoniec rozdelenie tried v testovanom dátovom súbore.

4.7.1.4 Dashboard

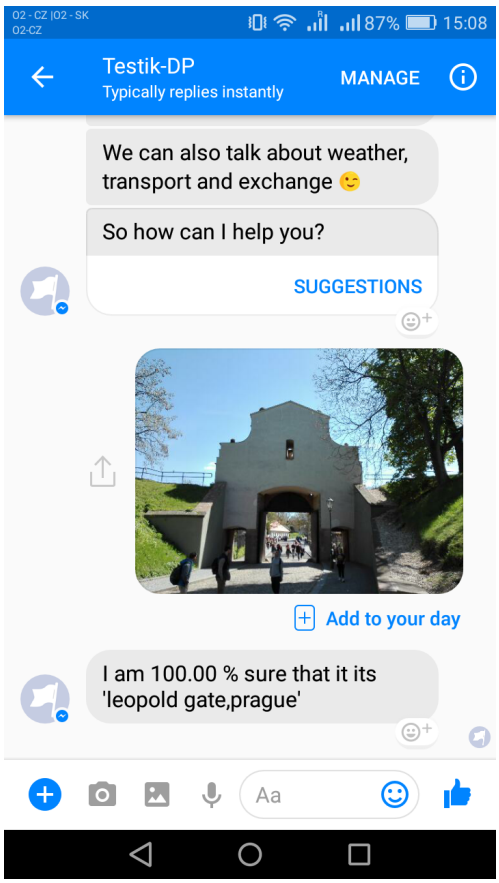
Ako súčasť svojej implementácie serveru som vytvoril taktiež jednoduchý web, ktorý slúži primárne na testovanie implementácie modelu a serveru, kontrolu výsledkov krížovej validácie, výsledkov validácie testovacích dátových súborov a zistenie informácií o použitom dátovom súbore. Tento web je písaný v jazyku *PHP*. Využíva frameworku *Nette* pre zjednodušenie práce. Výhodou takéhoto dashboardu je to, že umožňuje otestovať samotné rozhranie pre klasifikáciu, proces klasifikácie ako celok a napríklad nájsť prípady, ktoré klasifikátor ne-

priradí do správnej triedy a ani nezamietne. Ďalšou výhodou zobrazenie aktuálnych výsledkov validácie v lepšie čitateľnej forme pomocou grafov a tabuliek.

Tento dashboard je dostupný na adrese https://dp.daemon.mameweb.cz.

4.7.2 Golem

Druhou častou mojej integrácie bolo napojenie frameworku na chatbota *Golem*. Vzhľadom na mnou vytvorené jednoduché API bola táto úprava minimálna. Ukážku funkčnosti môžete vidiet na obrázku 4.30. Chatbot komunikuje s predikčným serverom priamo cez *Unix domain socket*, pričom ako požiadavok na klasifikáciu mu odosiela URL adresu na obrázok. V prípade, že zvolený obrázok je zle klasifikovaný, je uživatelovi zdelená informácia o tom, že obrázok sa nepodarilo klasifikovať. Test tejto integrácie prebehol cez uzavretú stránku, do ktorej je možný prístup len po schválení. Tento test som vykonal sám, päť krát na rôznych fotografiách.



Obr. 4.30: Ukážka rozpoznania pamiatky a zobrazenie v aplikácií *Messanger*

Záver

V tejto práci som analyzoval existujúce prístupy pri rozpoznávaní objektov z fotografií. Analyzoval ich jednotlivé súčasti a možnosť ich využitia pri rozpoznávaní vybraných historických budov v Prahe. Následne som vybral určité vhodné pamiatky v Prahe. Pre tieto som vytvoril súbor vstupných obrázkov. Dáta pochádzali z *Google Street View*, *Google Custom Search* a zo snímkov extrahovaných z videa. Ako vhodnú metódu som zvolil extrakciou príznakov z fotografií pomocou predtrénovaných hlbokých konvolučných sietí. Špecificky siete *Resnet-50*, ktorá sa javila ako najlepšia z testovaných. Nad touto konvolučnou sieťou som postavil štandardný viacvrstvový perceptron s jednou skrytou vrstvou, s 1 000 neurónmi v skrytej vrstve a *ReLU* aktivačnou funkciou. Výsledkom krížovej validácie tohto modelu s využitím *DenseNet-121* na predspracovanie bola 72.9 % *Accuracy*. Výsledky testov tohoto modelu využívajúceho sieť *ResNet-50* boli však úspešnejšie nakoľko preukázali úspešnosť okolo 85 % aj na upravených fotografiách. Z takto naučeného modelu som následne vytvoril jednoduchý framework s API. Tento framework mi umožnil testovať rôzne fotografie dostupné na internete a zobraziť výsledky validácie a informácie o použitom dátovom súbore. Framework som následne napojil na chatbota *Golem* vyvíjaného u nás na fakulte a otestoval celkovú integráciu, výsledkom, ktorej je jednoduchý spôsob rozpoznania historickej pamiatky pomocou mobilnej aplikácie *Messanger*.

Počas tejto práce som vytvoril deriváty pôvodných obrázkov. Jednalo sa o úpravu jasu, rotáciu, rozostrenie a zaostrenie obrázku. Týmito novými obrázkami vznikol súbor obrázkov, ktorý obsahoval skoro 2 milióny fotografií. Na základných súboroch obrázkov a ich extrahovaných príznakoch som otestoval rôzne modely mimo viacvrstvového perceptronu. Tieto modely sa však preukázali ako nedostatočné, prípadne ich trénovanie alebo klasifikácia boli príliš časovo náročné pre praktické využitie. Medzi týmito modelmi boli *K-NN*, *Rozhodovacie Stromy*, *Ensemble metódy Bagging* a *Boosting*. Tieto modely boli testované s rôznymi parametrami, výsledky týchto testov je možné nájsť buď v samotnej práci alebo na priloženom médiu. V práci som taktiež skúšal na-

učit hlboké konvolučne siete rozpoznávať vybrané pamiatky na vytvorených obrázkoch. Tieto konvolučné siete boli *AlexNet*, *Dvojitý AlexNet* a môj vlastný návrh vychádzajúci z architektúry *AlexNet*. Tieto siete, až na môj návrh, neboli schopné sa správne naučiť rozpoznávať zvolené pamiatky. Môj návrh toto dokázal, no vzhľadom na nedostatok času som sa rozhodol zvoliť model s predtrénovanou hlbokou konvolučnou sieťou a klasifikátorom ktorý využije jej výsledkov.

Celkovo túto prácu považujem za úspešnú, nakoľko sa mi podarilo úspešne vytvoriť model, ktorý je schopný klasifikovať pamiatky. Pokiaľ by sa mala táto práca rozvíjať ďalej, uberal by som sa smerom k využitiu vlastného návrhu konvolučnej siete. Tento model by som sa pokúsil minimalizovať a zrýchliť jeho učenie. Prípadne pridať ďalšie objekty a otestovať klasifikáciu s rôznymi fotoaparátmi za rôznych podmienok ako napríklad daždivé počasie, iné ročné obdobie a ďalšie.

Literatúra

[1] Crudge, A.; Thomas, W.; Zhu, K.: Landmark Recognition Using Machine Learning.

[2] Lowe, D. G.: Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, ročník 60, č. 2, 2004: s. 91–110.

[3] Bay, H.; Tuytelaars, T.; Van Gool, L.: Surf: Speeded up robust features. In *European conference on computer vision*, Springer, 2006, s. 404–417.

[4] Dalal, N.; Triggs, B.: Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, ročník 1, IEEE, 2005, s. 886–893.

[5] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, ročník abs/1409.1556, 2014.

[6] Viola, P.; Jones, M. J.: Robust real-time face detection. *International journal of computer vision*, ročník 57, č. 2, 2004: s. 137–154.

[7] Rischka, M.; Conrad, S.: Landmark Recognition: State-of-the-Art Methods in a Large-Scale Scenario. In *LWA*, 2014.

[8] Rischka, M.; Conrad, S.: Image Landmark Recognition with Hierarchical K-Means Tree. In *Datenbanksysteme für Business, Technologie und Web (BTW), 16. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme"(DBIS), 4.-6.3.2015 in Hamburg, Germany. Proceedings*, 2015, s. 455–464. Dostupné z: `http://subs.emis.de/LNI/Proceedings/Proceedings241/article16.html`

[9] Ren, S.; He, K.; Girshick, R. B.; aj.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, ročník abs/1506.01497, 2015. Dostupné z: `http://arxiv.org/abs/1506.01497`

[10] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, editace F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012, s. 1097–1105. Dostupné z: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`

[11] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. *CoRR*, ročník abs/1512.03385, 2015. Dostupné z: `http://arxiv.org/abs/1512.03385`

[12] Huang, G.; Liu, Z.; Weinberger, K. Q.: Densely Connected Convolutional Networks. *CoRR*, ročník abs/1608.06993, 2016. Dostupné z: `http://arxiv.org/abs/1608.06993`

[13] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org. Dostupné z: `http://tensorflow.org/`

[14] Jia, Y.; Shelhamer, E.; Donahue, J.; aj.: Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[15] Buitinck, L.; Louppe, G.; Blondel, M.; aj.: API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, s. 108–122.

[16] Kingma, D. P.; Ba, J.: Adam: A Method for Stochastic Optimization. *CoRR*, ročník abs/1412.6980, 2014. Dostupné z: `http://arxiv.org/abs/1412.6980`

[17] Tange, O.: GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine*, ročník 36, č. 1, Feb 2011: s. 42–47. Dostupné z: `http://www.gnu.org/s/parallel`

[18] Takeuchi, Y.; Hebert, M.: Evaluation of image-based landmark recognition techniques. *Robotics Institute*, 1998: str. 465.

[19] Li, Y.; Crandall, D. J.; Huttenlocher, D. P.: Landmark classification in large-scale image collections. In *Computer vision, 2009 IEEE 12th international conference on*, IEEE, 2009, s. 1957–1964.

[20] Li, X.; Wu, C.; Zach, C.; aj.: Modeling and recognition of landmark image collections using iconic scene graphs. In *European conference on computer vision*, Springer, 2008, s. 427–440.

[21] Zheng, Y.-T.; Zhao, M.; Song, Y.; aj.: Tour the world: building a web-scale landmark recognition engine. In *Computer vision and pattern recognition, 2009. CVPR 2009. IEEE conference on*, IEEE, 2009, s. 1085–1092.

[22] Erhan, D.; Szegedy, C.; Toshev, A.; aj.: Scalable Object Detection using Deep Neural Networks. *CoRR*, ročník abs/1312.2249, 2013. Dostupné z: `http://arxiv.org/abs/1312.2249`

[23] Girshick, R. B.; Donahue, J.; Darrell, T.; aj.: Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, ročník abs/1311.2524, 2013. Dostupné z: `http://arxiv.org/abs/1311.2524`

Dodatok **A**

Zoznam použitých skratiek

API Application programming interface

PCA Pricipal Components Analysis

KNN K-Nearest Neighbours

SVM Support Vector Machine

MLP Multilayer perceptrom

SIFT Scale-invariant feature transform

SURF Speeded-up Robust features

HOG Histogram of Oriented Gradients

DOG Difference of Gaussians

ROI Region of Interest

RPN Region Proposal Network

NLP Natural Language Processing

JSON JavaScript Object Notation

GPU Graphical processing unit

CSV Comma separated values

CUDA Compute Unified Device Architecture

SGD Stochastic Gradient Descent

URL Uniform Resource Locator

Dodatok **B**

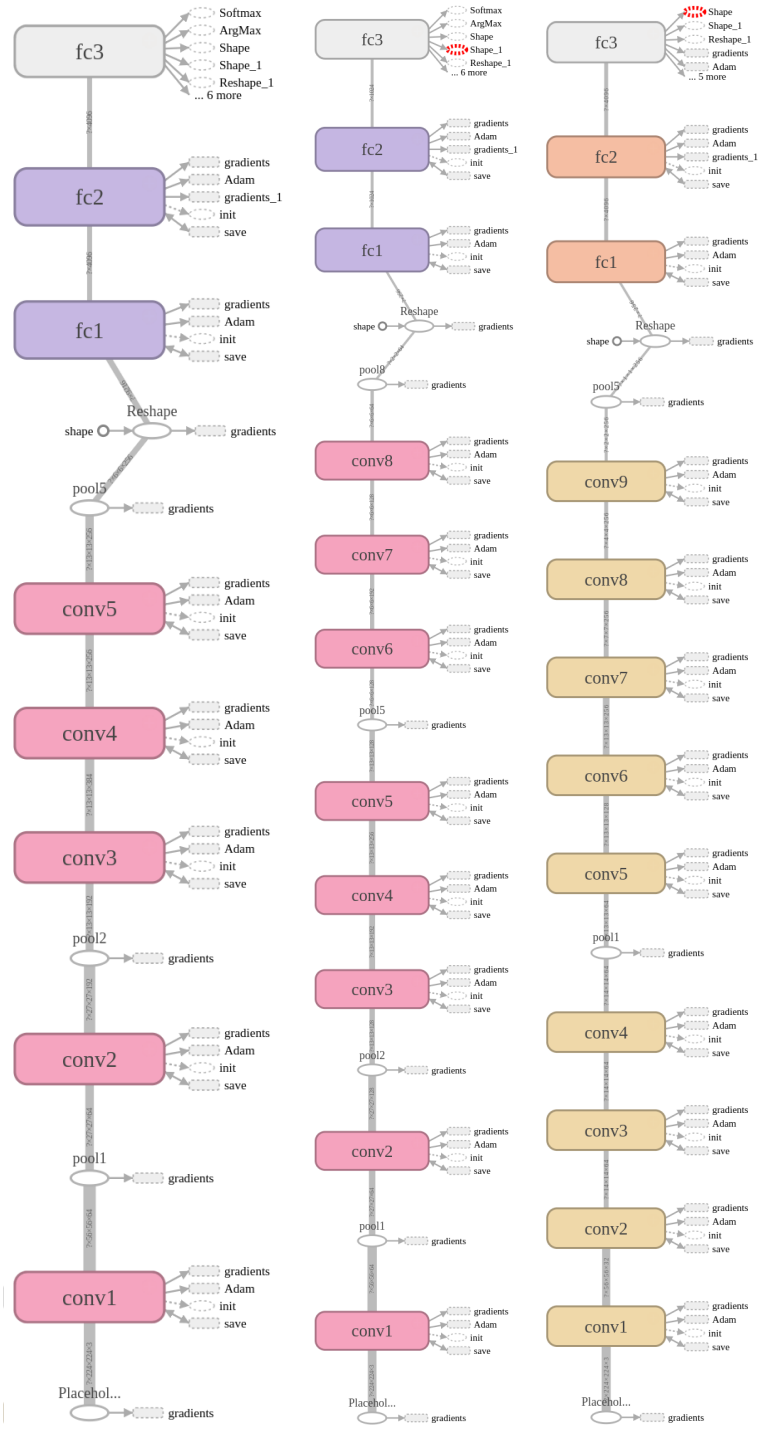
Obsah priloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF
  - `thesis.ps` ..... text práce vo formáte PS
- `measurements`
  - `summaries` ..... priebehy učenia sietí pre nástroj *TensorBoard*
  - `results` ..... výsledky validácií a meraní

Dodatok C

Štruktúra testovaných konvolučných sietí

Táto kapitola obsahuje popis štruktúry jednotlivých testovaných hlbokých sietí, popis pozostáva z exportu grafu z nástroja *TensorBoard* a popis jednotlivých vrstiev viditelných na týchto obrázkoch. Štruktúra bola vytvorená tak aby bolo možné tieto siete trénovať na grafických kartách *Tesla K20m*, dostupných na serveroch virtuálnej organizácie *MetaCentrum*.



Obr. C.1: Grafické znázornenie použitých sietí, z ľava, AlexNet, Dvojitý Alexnet, Môj návrh hlbokej konvolučnej siete

Node name	Layers	Parameters
input image	-	224 x 224 x 3
conv1	conv + bias + ReLU	Kernels: 32, Size: 5x5 Stride: 4x4 Padding: same
conv2	conv + bias + ReLU	Kernels: 64, Size: 5x5 Stride: 4x4 Padding: same
conv3, conv4	conv + bias + ReLU	Kernels: 64, Size: 5x5 Stride: 1x1 Padding: same
pool1	Max Pool	Kernel size: 2x2 Stride: 1x1 Padding: valid
conv5	conv + bias + ReLU	Kernels: 128, Size: 2x2 Stride: 1x1 Padding: same
conv6	conv + bias + ReLU	Kernels: 256, Size: 2x2 Stride: 1x1 Padding: same
conv7, conv8, conv9	conv + bias + ReLU	Kernels: 64, Size: 5x5 Stride: 1x1 Padding: same
pool5	Max Pool	Kernel size: 2x2 Stride: 2x2 Padding: valid
fc1, fc2, fc3	matmul + bias + ReLU	Hidden Layer Size: 4096

Tabuľka C.1: Parametre jednotlivých vrstiev v sieti *MyNetwork*

Node name	**Layers**	**Parameters**
input image	-	224 x 224 x 3
conv1	conv + bias + ReLU	Kernels: 64, Size: 11x11 Stride: 4x4 Padding: same
pool1	Max Pool	Kernel size: 3x3 Stride: 2x2 Padding: valid
conv2	conv + bias + ReLU	Kernels: 192, Size: 5x5 Stride: 1x1 Padding: same
pool2	Max Pool	Kernel size: 3x3 Stride: 2x2 Padding: valid
conv3	conv + bias + ReLU	Kernels: 384, Size: 3x3 Stride: 1x1 Padding: same
conv4	conv + bias + ReLU	Kernels: 256, Size: 3x3 Stride: 1x1 Padding: same
conv5	conv + bias + ReLU	Kernels: 256, Size: 3x3 Stride: 1x1 Padding: same
pool5	Max Pool	Kernel size: 3x3 Stride: 2x2 Padding: valid
fc1, fc2, fc3	matmul + bias + ReLU	Hidden Layer Size: 4096

Tabuľka C.2: Parametre jednotlivých vrstiev v sieti *AlexNet*

Node name	Layers	Parameters
input image	-	224 x 224 x 3
conv1	conv + bias + ReLU	Kernels: 64, Size: 11x11 Stride: 4x4 Padding: same
pool1	Max Pool	Kernel size: 3x3 Stride: 2x2 Padding: valid
conv2	conv + bias + ReLU	Kernels: 192, Size: 5x5 Stride: 1x1 Padding: same
pool2	Max Pool	Kernel size: 3x3 Stride: 2x2 Padding: valid
conv3	conv + bias + ReLU	Kernels: 384, Size: 3x3 Stride: 1x1 Padding: same
conv4	conv + bias + ReLU	Kernels: 256, Size: 3x3 Stride: 1x1 Padding: same
conv5	conv + bias + ReLU	Kernels: 256, Size: 3x3 Stride: 1x1 Padding: same
pool5	Max Pool	Kernel size: 3x3 Stride: 2x2 Padding: valid
conv6	conv + bias + ReLU	Kernels: 192, Size: 3x3 Stride: 1x1 Padding: same
conv7	conv + bias + ReLU	Kernels: 128, Size: 3x3 Stride: 1x1 Padding: same
conv8	conv + bias + ReLU	Kernels: 64, Size: 3x3 Stride: 1x1 Padding: same
pool8	Max Pool	Kernel size: 3x3 Stride: 2x2 Padding: valid
fc1, fc2, fc3	matmul + bias + ReLU	Hidden Layer Size: 4 096

Tabuľka C.3: Parametre jednotlivých vrstiev v sieti *Dvojitý AlexNet*

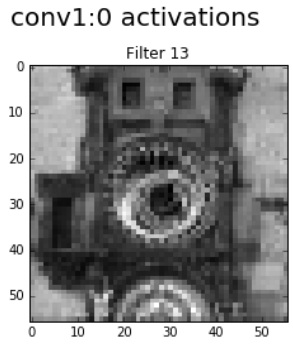
Dodatok D

Aktivácie v sieti *MyNetwork*

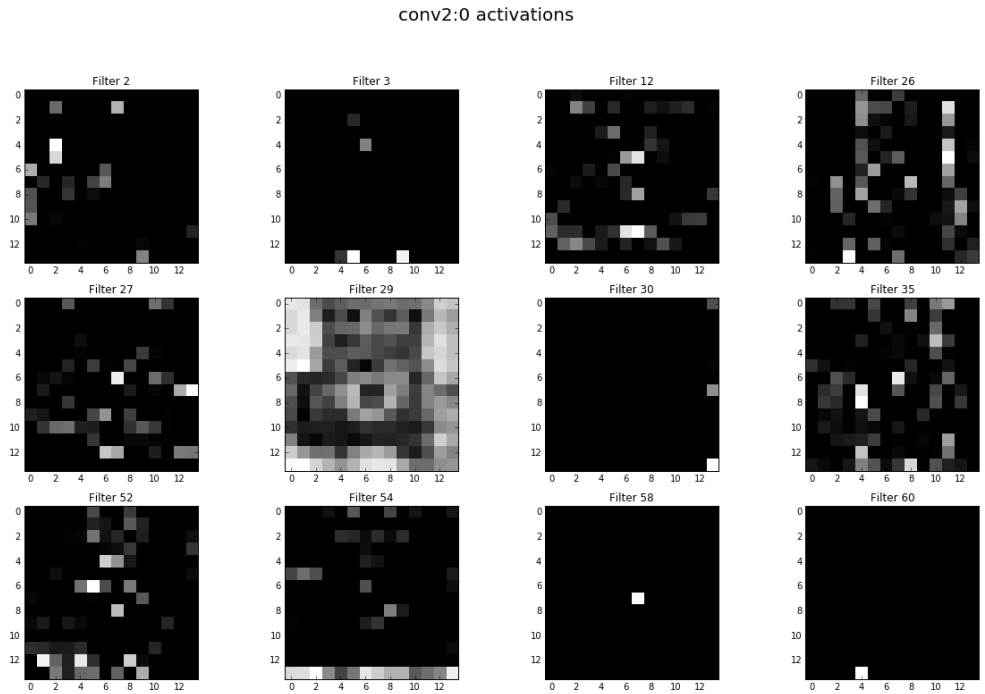
Táto kapitola obsahuje zobrazenie výstupov jednotlivých konvolučných a pooling vrstiev modelu *MyNetwork*, pre ukážkový vstup D. Obrázky výstupov obsahujú len kanály (výstupy pre jednotlivé kernely), ktoré obsahujú aj inú hodnotu ako 0.

Táto sieť bola natrénovaná na snímkoch z videa a má 95 % úspešnosť. Vstupný obrázok je stiahnutý z internetu a nebol súčasťou trénovacích dát.

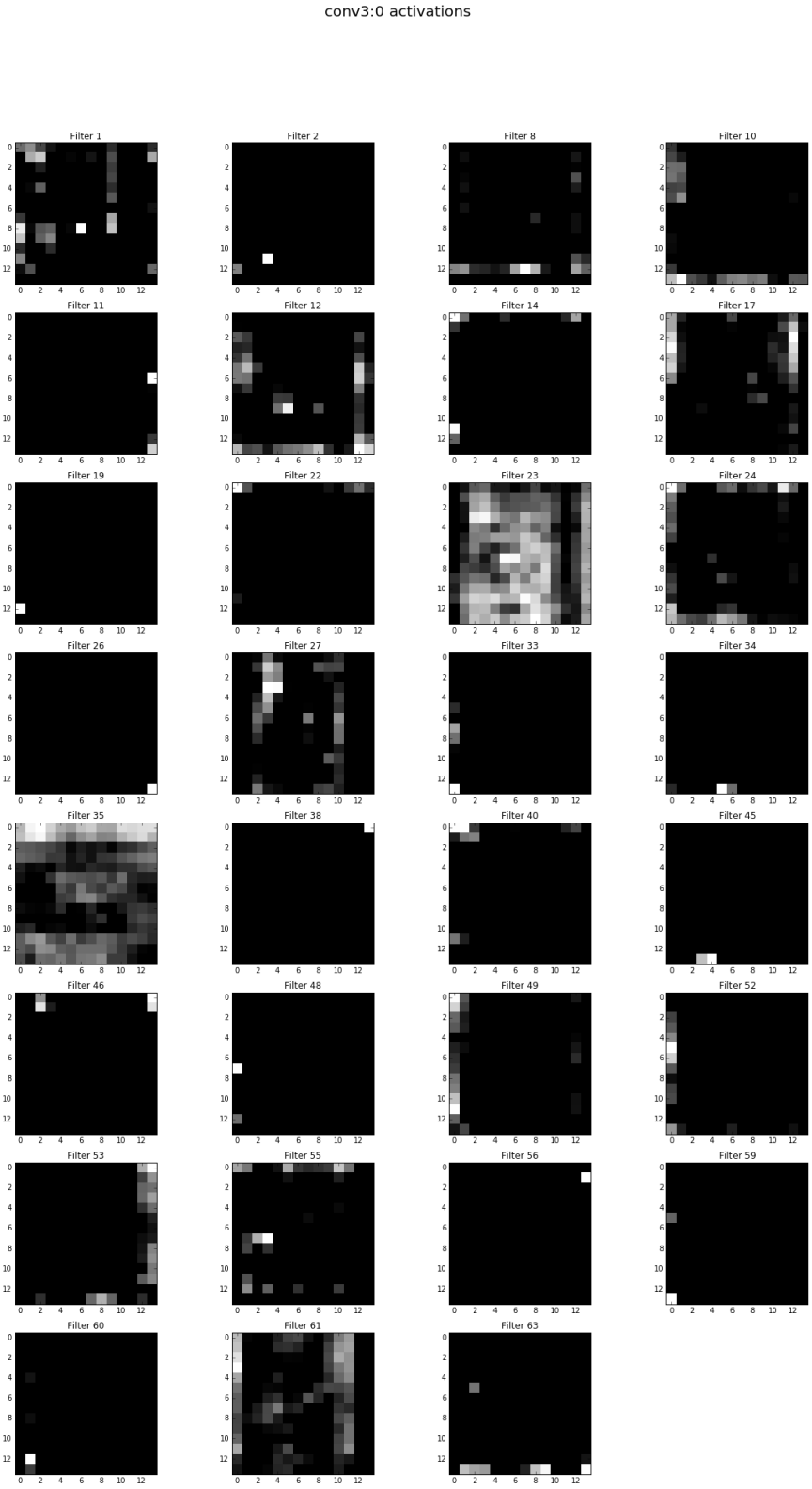
Obr. D.1: Ukážkový vstup, pre obráky aktivácií vrstiev modelu *MyNetwork*



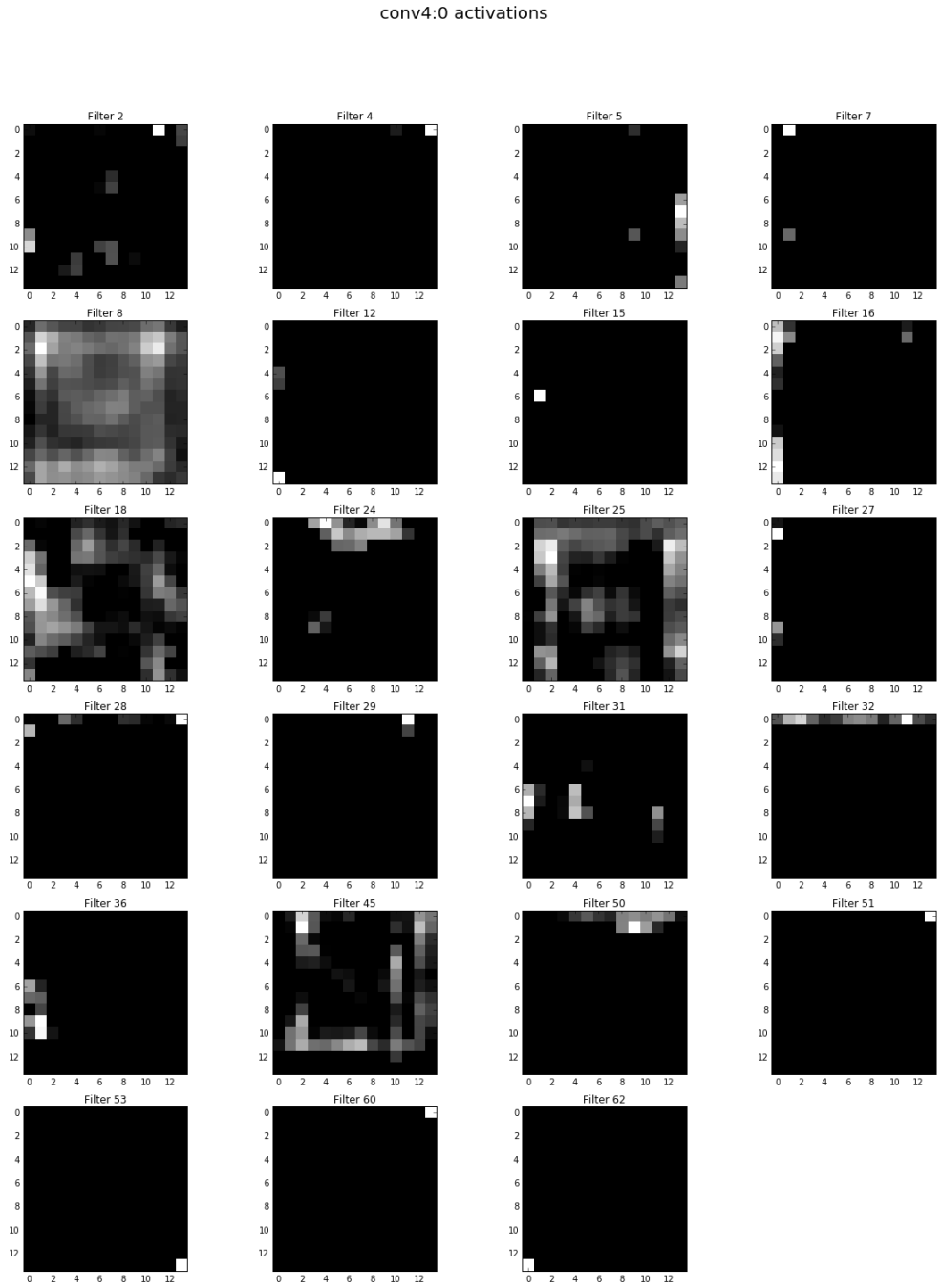
Obr. D.2: Zobrazenie výstupu vrstvy *conv1* v sieti *MyNetwork* pre ukážkový vstup



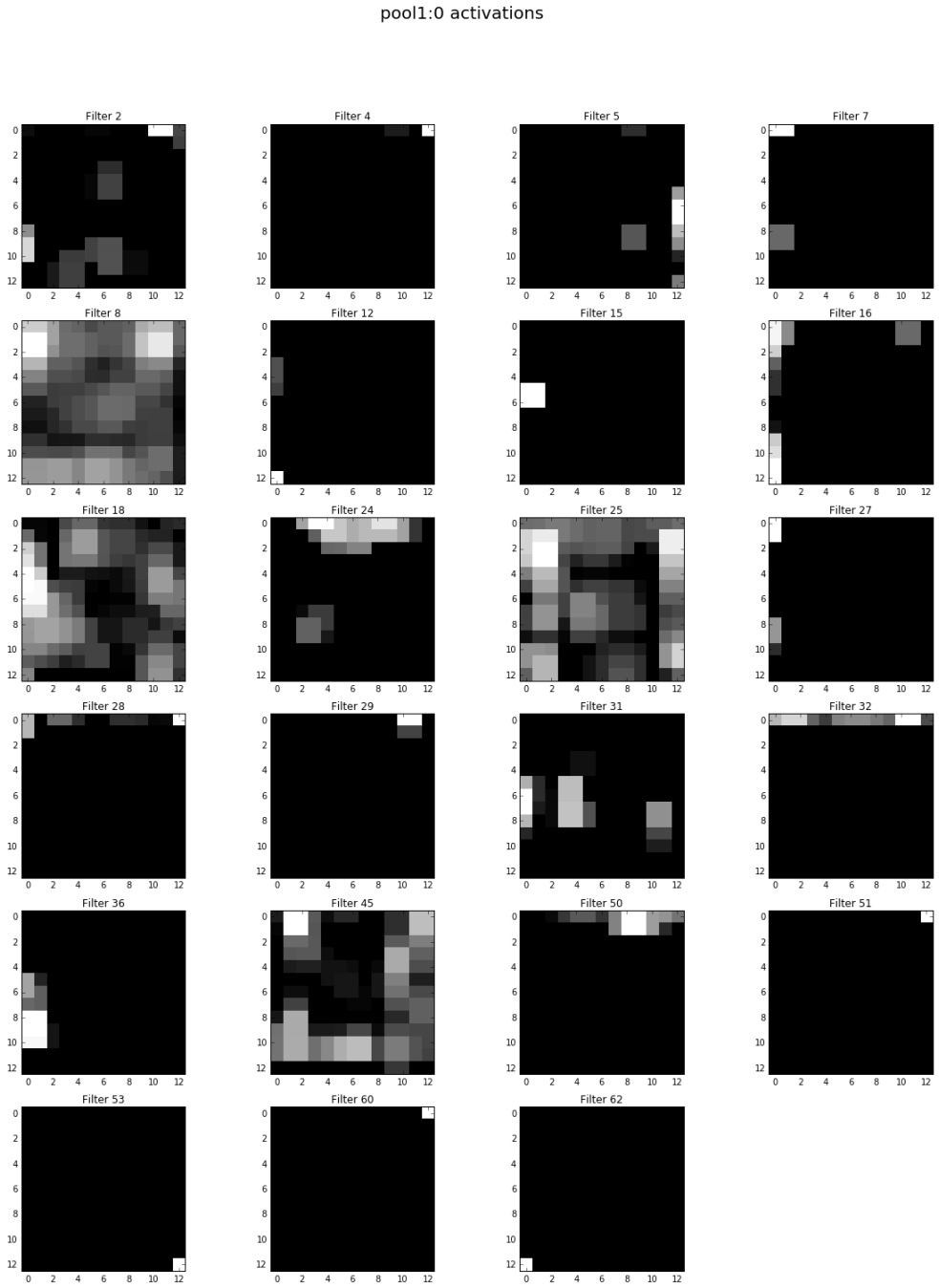
Obr. D.3: Zobrazenie výstupu vrstvy *conv2* v sieti *MyNetwork* pre ukážkový vstup



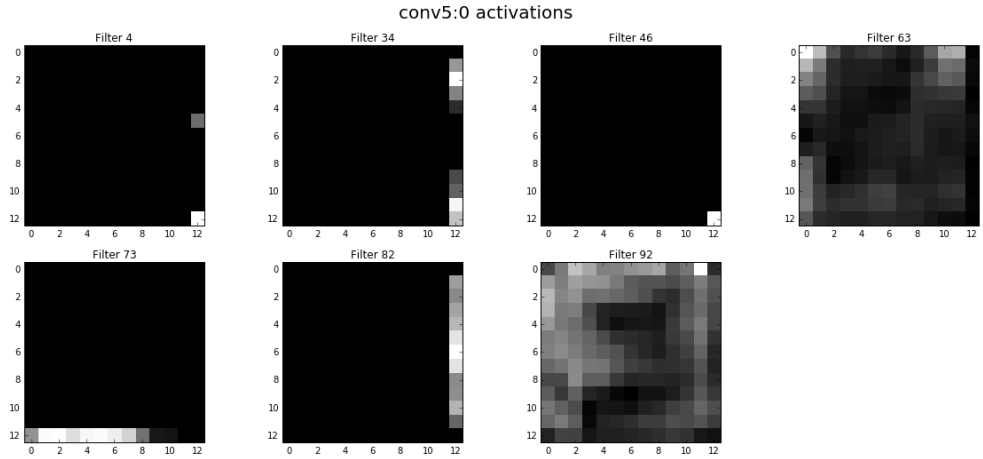
Obr. D.4: Zobrazenie výstupu vrstvy *conv3* v sieti *MyNetwork* pre ukážkový vstup



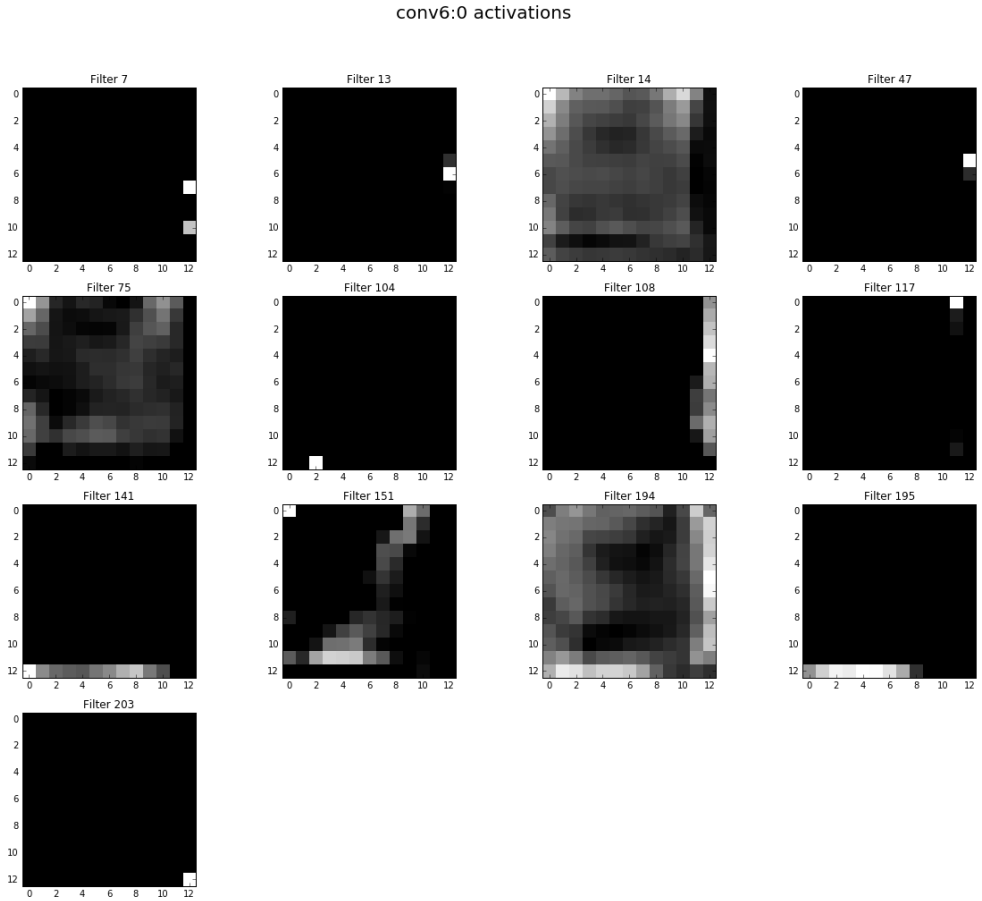
Obr. D.5: Zobrazenie výstupu vrstvy *conv4* v sieti *MyNetwork* pre ukážkový vstup



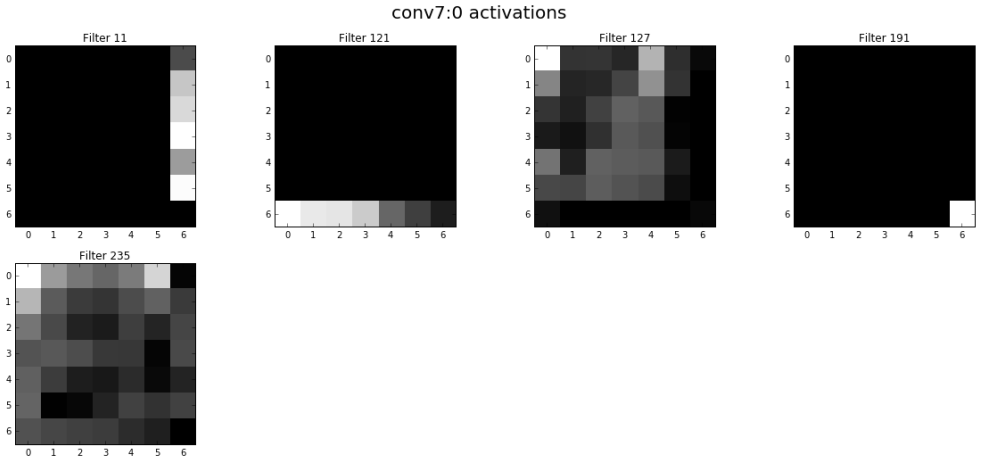
Obr. D.6: Zobrazenie výstupu vrstvy *pool1* v sieti *MyNetwork* pre ukážkový vstup



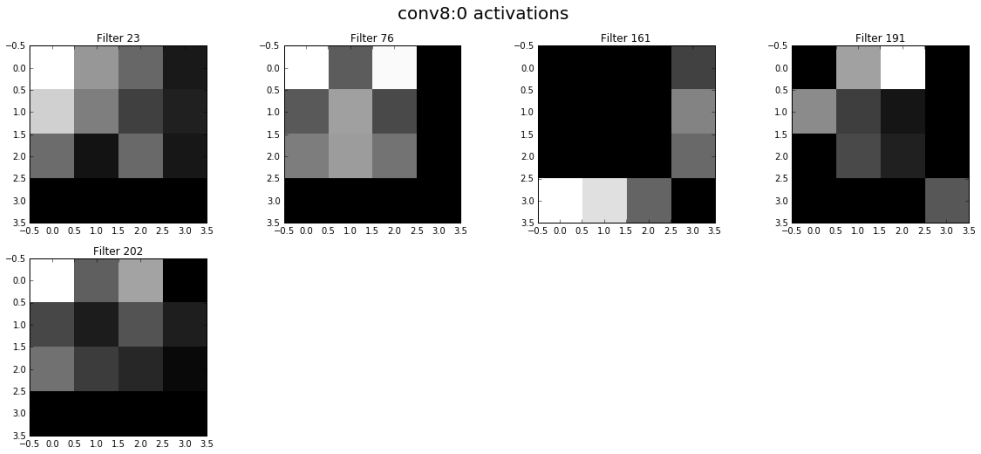
Obr. D.7: Zobrazenie výstupu vrstvy *conv5* v sieti *MyNetwork* pre ukážkový vstup



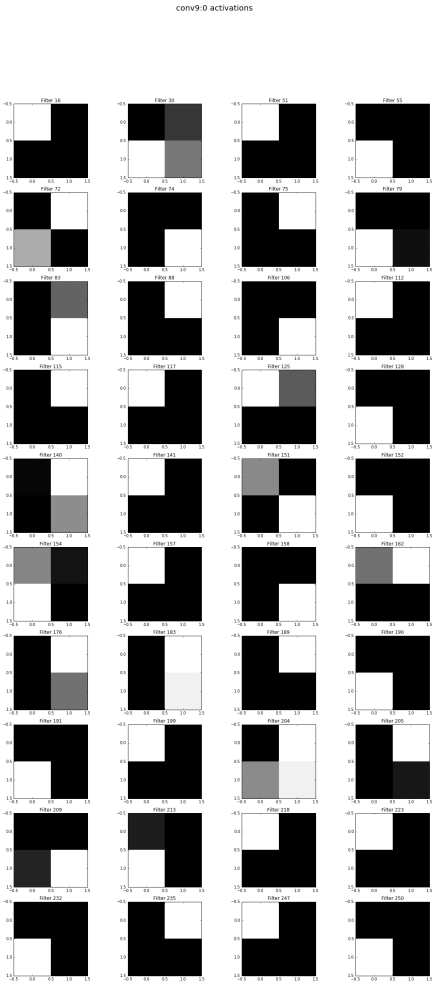
Obr. D.8: Zobrazenie výstupu vrstvy *conv6* v sieti *MyNetwork* pre ukážkový vstup



Obr. D.9: Zobrazenie výstupu vrstvy *conv7* v sieti *MyNetwork* pre ukážkový vstup



Obr. D.10: Zobrazenie výstupu vrstvy *conv8* v sieti *MyNetwork* pre ukážkový vstup



Obr. D.11: Zobrazenie výstupu vrstvy *conv9* v sieti *MyNetwork* pre ukážkový vstup

Dodatok E

Štruktúry výsledkov testov

Vzhľadom na počet vykonaných testov a textový rozsah tejto práce, som sa rozhodol uviesť len zlomok výsledkov. Uvedené výsledky som volil tak aby boli dostatočne reprezentatívne a vypovedajúce. Celkovo som uložil výsledky 1194 výsledkov testov vo formáte *JSON*, štruktúra výsledkov testov je relatívne jednoduchá, základom je pole záznamov výsledkov, počet týchto záznamov je zväčša 25 a viac, no môžu sa vyskytnúť testy v ktorých je len jeden. To z toho dôvodu že boli časovo náročné a tak boli rozdelené do viacerých úloh.

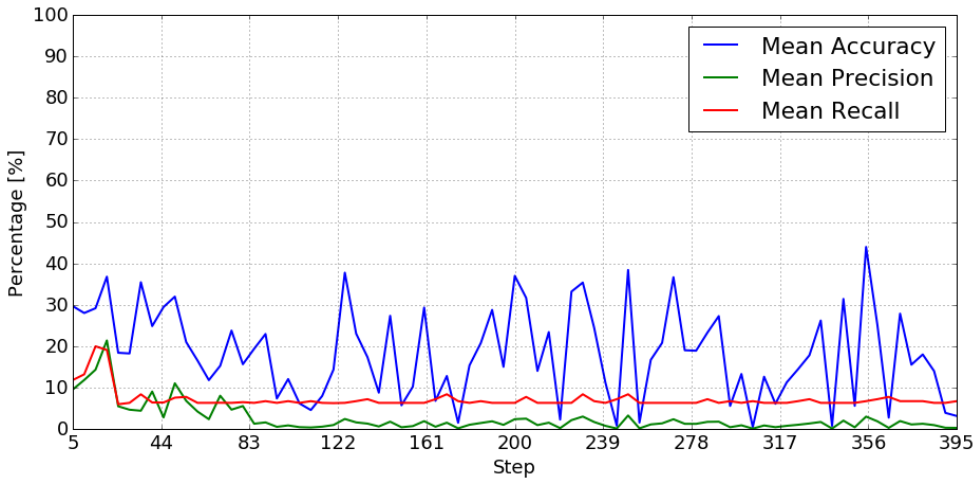
Samotný záznam výsledku je objekt, obsahujúci minimálne klúče *params* a *predictions*. Klúč *params* obsahuje parametre modelu pre tento výsledok. Kluč *predictions* obsahuje záznami z jednotlivých krížových validácií. Každý záznam zas obsahuje páry reálnych hodnôt a predikovaných. Z takto uložených výsledkov je možné vypočítať základné metriky používané v tejto práci.

Ďalšími možnými klúčami v zázname výsledku sú *metrics* a *mean_metrics*, tieto obsahujú metriky z uložených dát a ich primery.

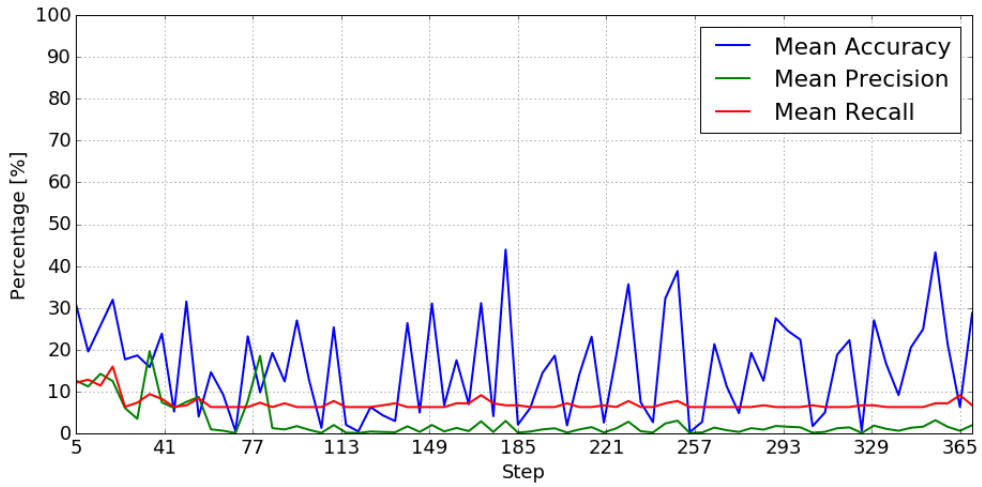
Na priloženom médiu sa nachádzajú skomprimované výsledky, celé rozbalené majú veľkost približne 20 GB.

Dodatok F

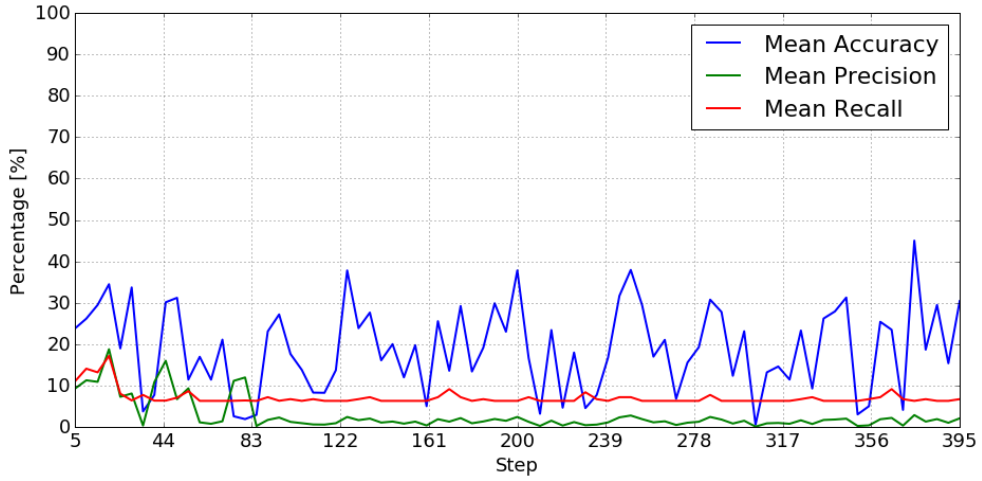
Priebehy učenia pri krížovej validácií



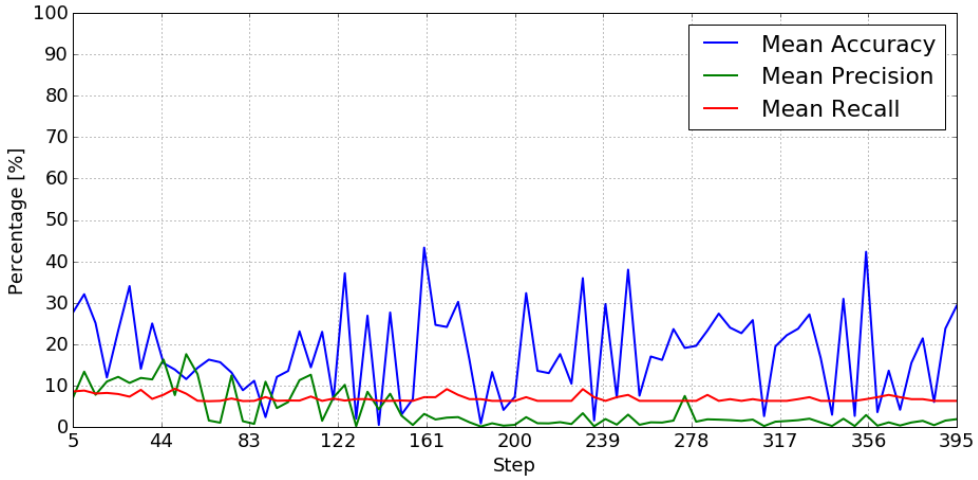
Obr. F.1: Priebeh učenia krížovej validácie klasifikátoru *MyNetwork* pri prvom preložení



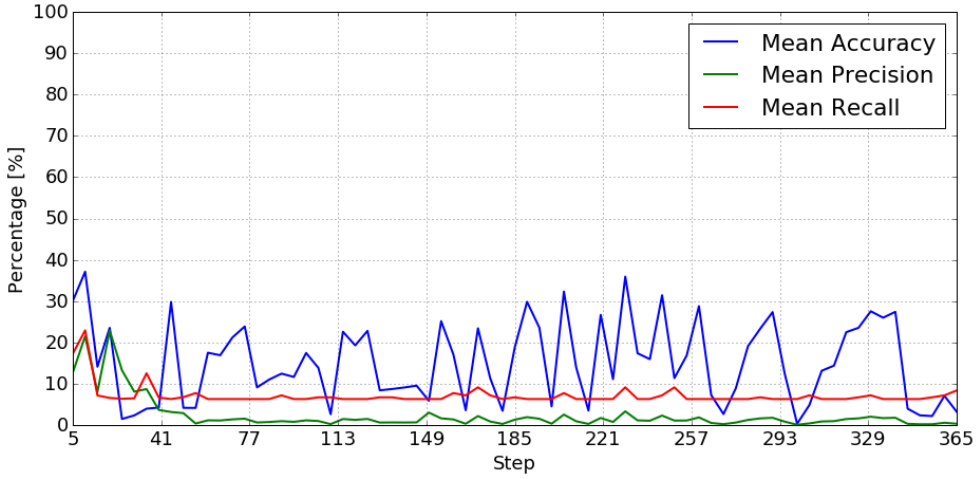
Obr. F.2: Priebeh učenia krížovej validácie klasifikátoru *MyNetwork* pri druhom preložení



Obr. F.3: Priebeh učenia krížovej validácie klasifikátoru *MyNetwork* pri treťom preložení



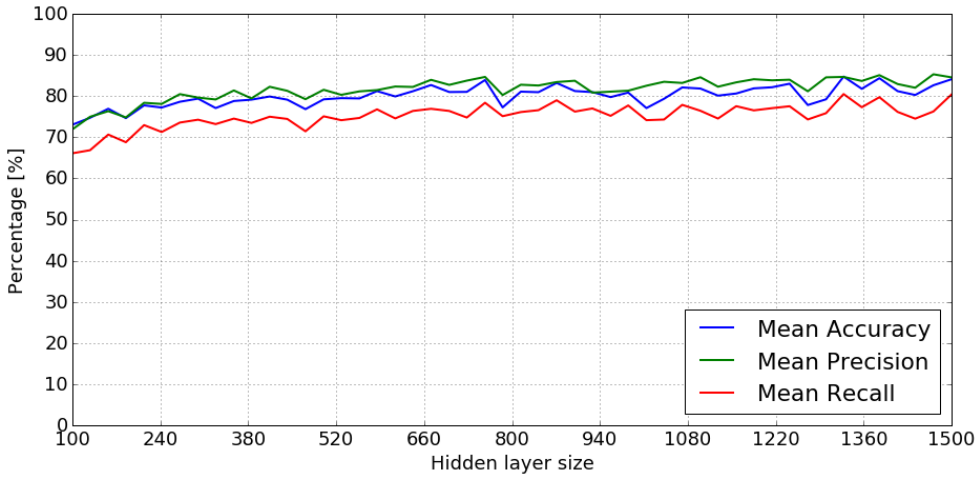
Obr. F.4: Priebeh učenia krížovej validácie klasifikátoru *MyNetwork* pri štvrtom preložení



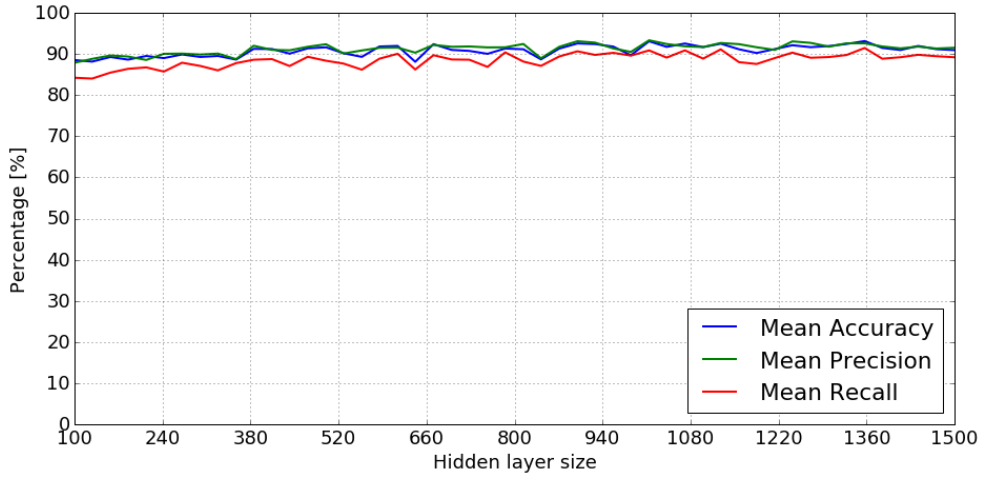
Obr. F.5: Priebeh učenia krížovej validácie klasifikátoru *MyNetwork* pri piatom preložení

Dodatok **G**

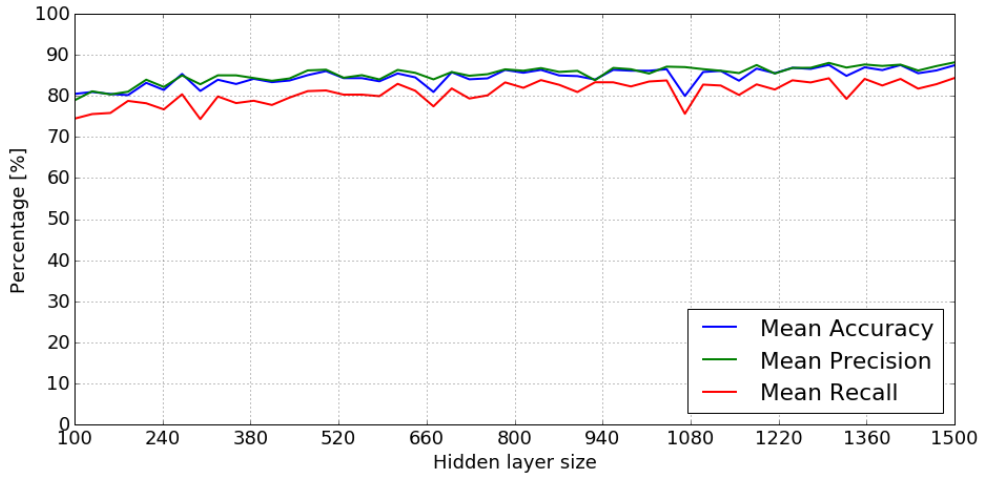
Merania trénovacích súborov



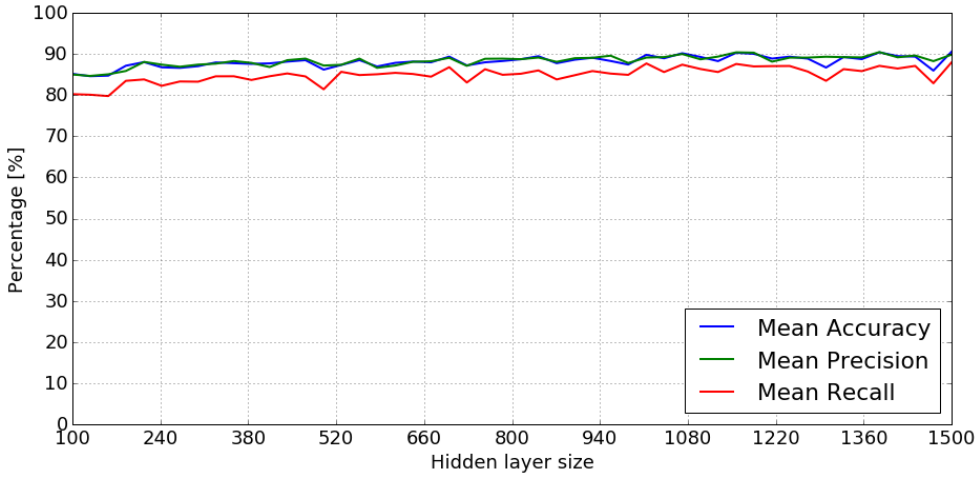
Obr. G.1: Výsledky klasifikátori *MLP* pre rôzne veľkosti skrytej vrstvy na obrázkoch z *Google Street View* s príznakmi extrahovanými pomocou *DenseNet-121*



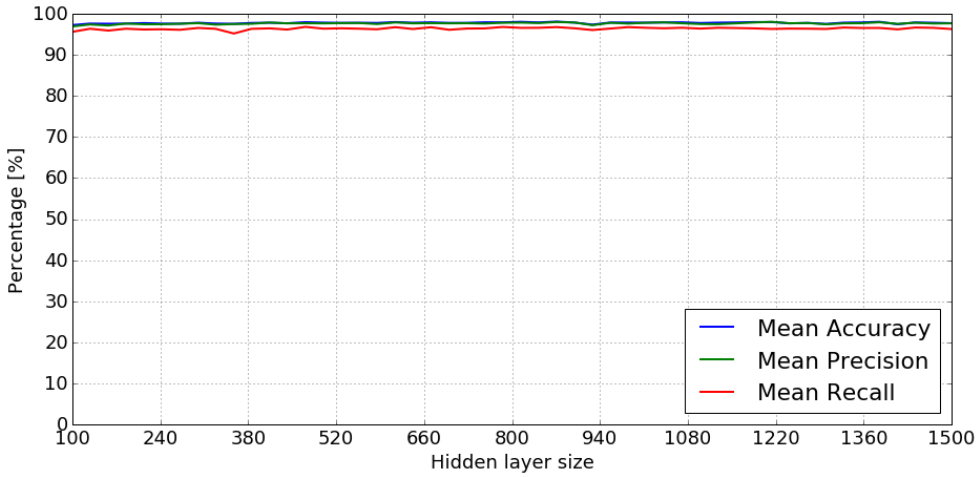
Obr. G.2: Výsledky klasifikátori *MLP* pre rôzne veľkosti skrytej vrstvy na obrázkoch z *Google Street View* s príznakmi extrahovanými pomocou *DenseNet-161*



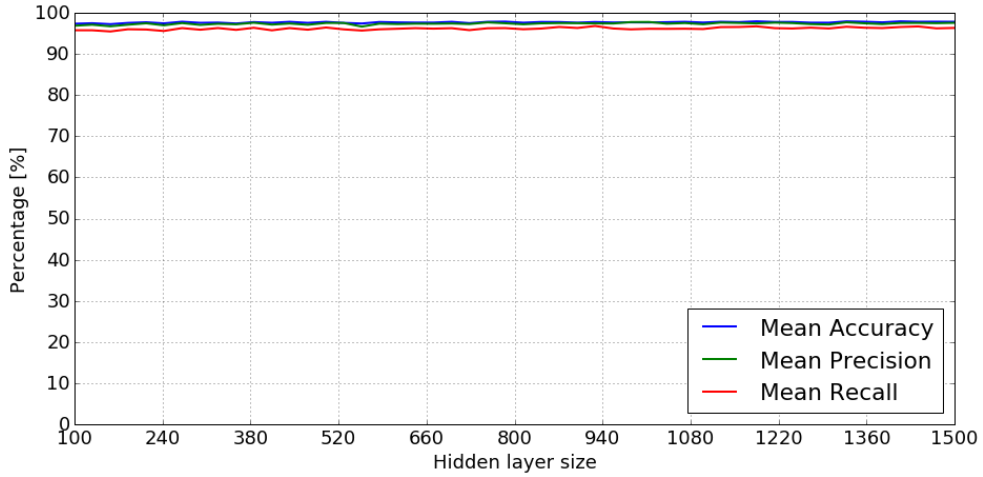
Obr. G.3: Výsledky klasifikátori *MLP* pre rôzne veľkosti skrytej vrstvy na obrázkoch z *Google Street View* s príznakmi extrahovanými pomocou *DenseNet-169*



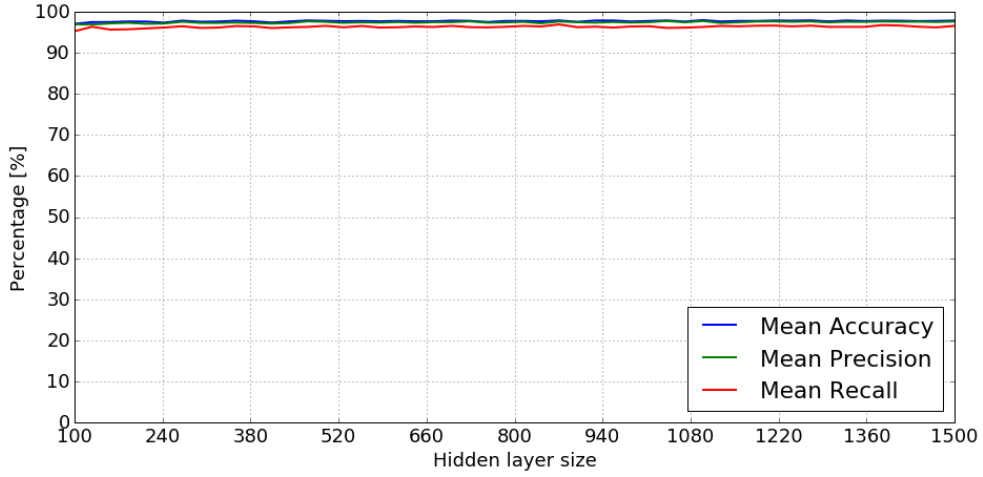
Obr. G.4: Výsledky klasifikátori *MLP* pre rôzne veľkosti skrytej vrstvy na obrázkoch z *Google Street View* s príznakmi extrahovanými pomocou *DenseNet-201*



Obr. G.5: Výsledky klasifikátori *MLP* pre rôzne veľkosti skrytej vrstvy na obrázkoch z *Google Street View* s príznakmi extrahovanými pomocou *ResNet-50*



Obr. G.6: Výsledky klasifikátori *MLP* pre rôzne veľkosti skrytej vrstvy na obrázkoch z *Google Street View* s príznakmi extrahovanými pomocou *ResNet-101*



Obr. G.7: Výsledky klasifikátori *MLP* pre rôzne veľkosti skrytej vrstvy na obrázkoch z *Google Street View* s príznakmi extrahovanými pomocou *ResNet-152*