VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE OBJEKTŮ POMOCÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ

DEEP LEARNING FOR OBJECT DETECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ANDREJ PANÍČEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LUKÁŠ TEUER**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Paníček Andrej**
Program: Informační technologie
Název: **Detekce objektů pomocí hlubokých neuronových sítí**
Deep Learning for Object Detection
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte základy neuronových sítí a back-propagation.
2. Vytvořte si přehled o současných metodách pro tvorbu hlubokých sítí, a hlavně konvolučních sítí.
3. Vyberte konkrétní metodu aplikovatelnou na detekci objektů.
4. Obstarejte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručný plakát prezentující vaši práci, její cíle a výsledky.

Literatura:

- Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- Grishick et. al.: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR 2014.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Teuer Lukáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Táto práca sa zaoberá detekciou objektov pomocou hlbokých neurónových sietí. V rámci riešenia som upravil, implementoval a natrénoval dobre známy model kaskádových neurónových sietí MTCNN tak aby dokázal vykonávať detekciu dopravných značiek. Trénovacie dáta boli vygenerované z dátových sád GTSRB a GTSDB. MTCNN ukázal solídny výkon na vyhodnocovacích dátach z dátovej sady GTSDB, kde dosiahol presnosť detekcie 97.8 %.

Abstract

This work deals with the object detection using deep neural networks. As part of the solution, I modified, implemented and trained the well-known model of cascade neural networks MTCNN so that it could perform the detection of traffic signs. The training data was generated from GTSRB and GTSDB data sets. MTCNN showed solid performance on the evaluation data, where the detection accuracy reached 97.8 %.

Kľúčové slová

neurón, hlboké neurónové siete, konvolučné neurónové siete, strojové učenie, umelá inteligencia, detekcia, MTCNN, detekcia značiek, GTSBD, GTSRB

Keywords

neuron, deep neural network, convolutional neural network, machine learning, artificial intelligence, detection, MTCNN, traffic sign detection, GTSBD, GTSRB

Citácia

PANÍČEK, Andrej. *Detekce objektů pomocí hlubokých neuronových sítí.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Lukáš Teuer

Detekce objektů pomocí hlubokých neuronových sítí

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Lukáša Teuera. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....
Andrej Paníček
16. mája 2019

Poďakovanie

Moja vďaka patrí vedúcemu Ing. Lukášovi Teuerovi, za poskytnutý čas, informácie a rady. Táto práca vznikla za podpory projektov CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

Obsah

Kapitola 1

Úvod

Momentálne žijeme v dobe asi najväčšieho rozmachu v obore neurónových sietí od jeho vzniku v 40. rokoch minulého storočia. Veľkú zásluhu na tom má hlavne obrovský technologické pokrok, ak si pre porovnanie vezmeme napríklad 90. roky, kedy prebiehala v poradí druhá fáza rozkvetu tohto oboru. Vtedy sa všetko pohybovalo skôr v teoretických medziach, pretože výskum bol limitovaný výpočtovými zdrojmi a malým obnosom ľahko dostupných dát. Od vtedy sa všetko posunulo dopredu s vývojom internetu a dodržiavaním Moorovho zákona[1], sme dospeli do bodu, kde hocikto s prístupom k internetu á prístup k obrovským obnosom dát behom pár klikov. Taktiež trénovanie neurónových sietí je dnes už relatívne dostupné aj z domu, bez použitia superpočítačov. Toto malo za následok, že technológie založené na neurónových sieťach sa stávajú súčasťou čoraz viac odvetví priemyslu. Každý rok prichádzajú nové a lepšie riešenia.

Jedno z odvetví, v ktorom neurónové siete excelujú je detekcia objektov z obrázkov a videí. Môžeme si spomenúť niektoré využitia, akými sú napríklad detekcia poznávacích značiek, tvárí, aut, ale aj detektory pre veľkú množinu tried. Táto technológia už dávno predčila človeka, a v súčasnosti je hlavným cieľom získať čo najväčšiu presnosť a zároveň si zachovať výkon v reálnom čase. Úspešnosť závisí vždy hlavne od vhodne navrhnutej architektúry.

Ja v tejto práci beriem dobre známy kaskádový model neurónových sietí MTCNN a s malou úpravou, ktorá mi umožní vykonať detekciu značiek ho implementujem v jazyku Python. Tento model je zložený z 3 jednoduchých neurónových sietí pre trénovanie, ktorých som použil knižnicu Pytorch.

V druhej kapitole 2 čitateľa oboznamujem so základmi hlbokých neurónových sietí, princípom ich učenia a v neposlednej rade popisom konvolučných sietí. Tieto poznatky uľahčujú pochopenie nasledujúcej kapitoly 3, kde popisujem detekčný model MTCNN, jeho architektúru a princíp detekcie tvárí. Štvrtá kapitola 4 popisuje všetko ohľadom použitých dátových sád prevzatých z internetu, a ako z nich generovať dátové sady validné pre účeli tréningu. Kapitola 5 zahŕňa informácie k použitým technológiám, úpravu architektúry modelu MTCNN a implementáciu podstatných častí tejto práce. Predposledná kapitola 6 je azda najdôležitejšia, popisuje tréning a experimentovanie, tu sú obsiahnuté všetky výsledky nadobudnuté v rámci tejto práce.

[1] https://www.investopedia.com/terms/m/mooreslaw.asp

Kapitola 2

Neurónové sieťe

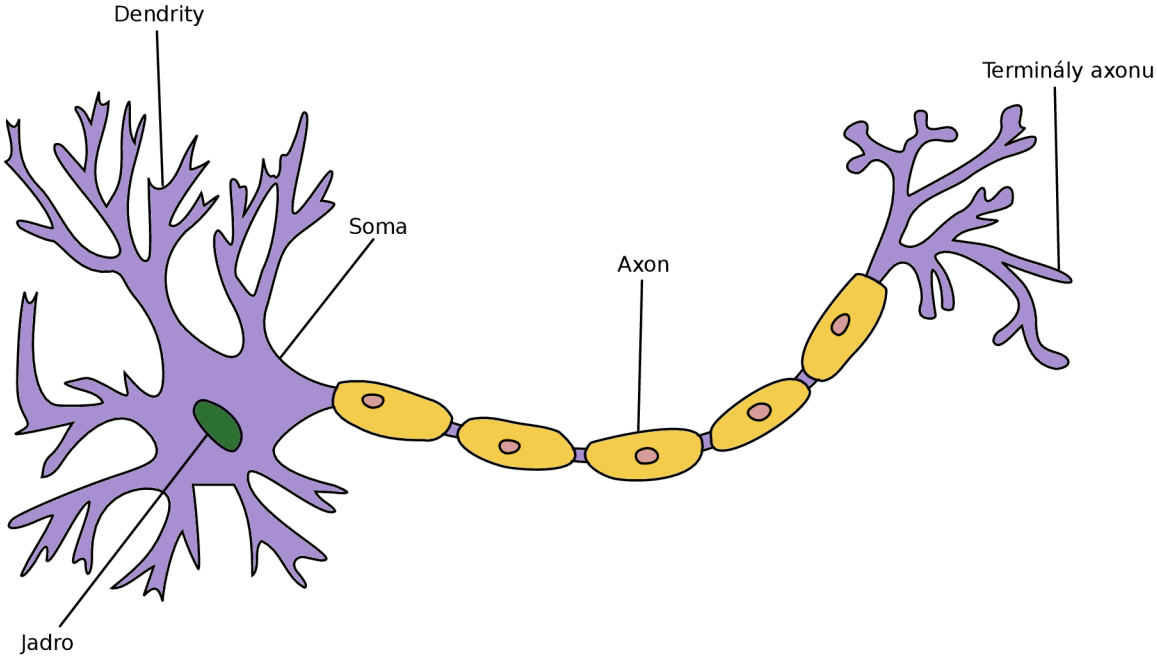
V tejto kapitole popisujem základné poznatky z oboru neurónových sietí, potrebné k plnému pochopeniu riešeného problému. Najskôr v sekcii 2.1, rozoberám základný princíp fungovania a prenosu signálov v neurónovej sústave pomocou biologických neurónov. Na to naväzujem v ďalšej časti 2.2, kde popisujem matematický modelu neurónu. V sekcii 2.3 vyzdvihujem prepojenie jednotlivých neurónov do veľkých celkov tzv. neurónových sietí. Súčasťou tejto sekcie je aj vysvetlenie základných metód a algoritmov vrátane algoritmu spätného šírenia chyby 2.3.2, potrebných k učeniu neurónových sietí. Posledná časť tejto kapitoly 2.4 je zameraná na špeciálny typ neurónových sietí, tzv. konvolučných neurónových sietí, používaných pri úlohách z oblasti počítačového videnia, akými sú klasifikácia alebo detekcia.

2.1 Biologický neuron

Než si predstavíme matematický model neurónu, popíšem jeho biologický vzor, na ktorom sa principiálne zakladá. Obecne môžeme povedať, že neurón je základnou jednotkou nervovej sústavy. Jednotlivé neuróny sú medzi sebou veľmi nahusto pospájané a tým tvoria obrovskú sieť. Počet prepojení sa neustále mení, nové vznikajú i zanikajú po celý život. Môžeme si to predstaviť tak, že keď sa človek niečo nové naučí tak je to spôsobené práve novo vzniknutými cestami medzi neurónmi. Naopak prerušením spojení je spôsobené zabúdanie. [16].

Na obrázku 2.1 je zobrazená štruktúru jednobunkového tela neurónu, na ktorej popíšem prenos signálu. Vstupné signály sú cez jednotlivé dendrity privedené do tela neurónu, pomenované „soma“. V tele sa nachádza jadro, v ktorom sa hodnoty signálov, sčítajú a v prípade, že výsledok sumy vstupných hodnôt dosiahne určitého prahu, neurón je aktivovaný a generuje signál ďalej. V tom prípade je následne vyvedení z tela cez axon, ktorý sa eventuálne rozvetví do terminálov axonu. Tie z terminálov, ktoré sú pripojené k ďalším neurónom pokračujú v šírení signálu cez dendrity. Miesto kde dochádza k prepojeniu medzi jednotlivými neurónmi sa nazýva synapsia. Každá synapsia má určitú váhu, ovplyvňujúcu silu prechádzajúceho signálu. Po každom prechode sa hodnoty váhy adaptujú [8, 16]. Synaptické prepojenia môžeme rozdeliť na:

- exitačné - zosilňujú prechádzajúci signál
- inhibičné - tlmia prechádzajúci signál



Obr. 2.1: Telo biologického neurónu. Obrázok je upravený, originál prevzatý z [5].

2.2 Matematický model neuronu

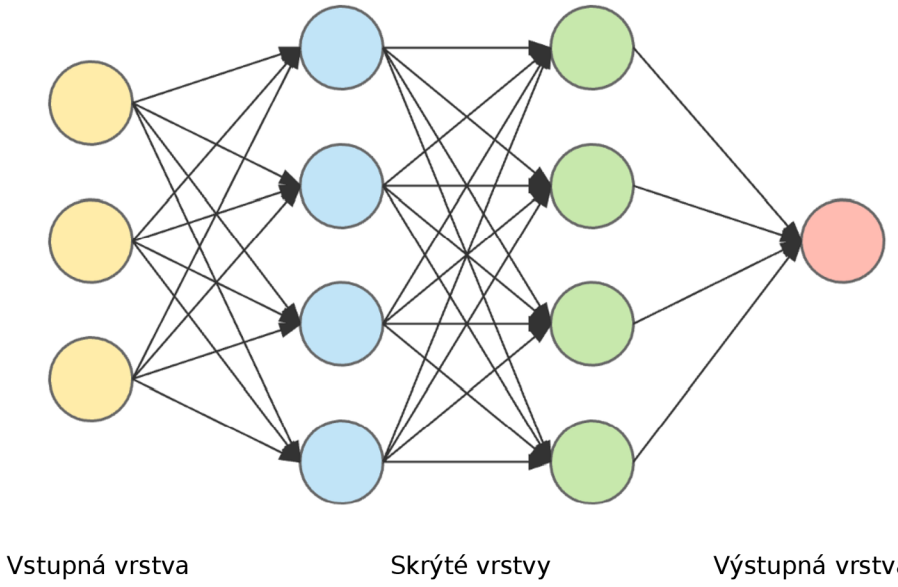
Matematický model pozostáva z $n_1, .., n_n$ reálnych vstupov tie predstavujú dendrity. Rovnako ako u biologického modelu sú vstupy ohodnocované váhami $w_1..w_n$, kde w je taktiež reálne číslo. Do neurónu vstupuje ešte jedna hodnota a tou je prah p. Sumou ohodnotených vstupov a prahovej hodnoty, získame vnútorný potenciál neurónu O. Výsledná hodnota potenciálu je propagovaná na vstup nelineárnej aktivačnej funkcie f, ktorá rozhoduje o tom či bude daný neurón aktivovaný a podľa typu aktivačnej funkcie, hodnotu výstupu šíreného ďalej. Pre matematický zápis funkcie neurónu viď 2.1 [16].

$$y = f(O) = f(\sum_{i=0}^{n}(x_i w_i) + p) \tag{2.1}$$

2.3 Hlboké neurónové siete

Používa sa aj pomenovanie dopredné neurónové siete. Zakladajú sa na rovnakej myšlienke ako biologické neuróny v nervovej sústave, spájaním neurónov do väčších celkou s cieľom získať schopnosť riešenia komplexných úloh. Uvážime, že jeden neurón dokáže aproximovať jednoduchú funkciu, prepojením v kombinácii s ďalšími neurónmi získame aproximáciu zloženej funkcie tzn. čím viac neurónov za sebou napojíme, tým zložitejší problém dokáže sieť riešiť. Neuróny sú samy o sebe iba funkcie s náhodne nastavenými váhami, ktoré ohodnocujú vstupy. Využitie z nich je možné získať až tréningom, kde je cez sieť propagované obrovské množstvo informácií, pre ktoré poznáme správne výsledky. V prípade, že sa predikcia výsledkov siete líši od tých očakávaných, adaptujeme váhy. Trénovanie siete teda v podstate predstavuje optimalizáciu váhových koeficientov každého prepojenia.

Časti bežnej neurónovej siete, môžeme vidieť na obrázku 2.2, obsahuje niekoľko takzvaných skrytých vrstiev a každá takáto vrstve pozostáva z niekoľkých na sebe nezávislých neurónov medzi ktorými neprebieha žiadna komunikácia. Okrem skrytých vrstiev ďalej obsahuje jednu vstupnú a jednu výstupnú vrstvu. Počet skrytých vrstiev určuje hĺbku siete, z toho vychádza aj pomenovanie hlboké neurónové siete [3].



Obr. 2.2: Jednoduchý model neurónovej siete s dvomi skrytými vrstvami. Obrázok je prevzatý z [2].

Ak je výstup každého neurónu z jednej skrytej vrstvy napojený na vstup každého neurón v nasledujúcej vrstve, hovoríme o nej že je plne prepojená. Plne prepojená vrstva je najbežnejší typ skrytej vrstvy, poznáme ale aj iné, napríklad konvolučná vrstva, ktorej sa bližšie venujem v sekcii 2.4, tá je používaná pri spracovaní obrázkových dát.

Pri riešení konkrétnej úlohy nastáva otázka, koľko skrytých vrstiev použiť, koľko neurónov v každej vrstve, akú aktivačnú funkciu atď. Tieto otázky nemajú jasnú odpoveď keďže neexistuje univerzálny návrh, ktorý by fungoval pre všetky úlohy. Avšak existujú zaužívané techniky, ktoré jednému môžu návrh dosť zjednodušiť.

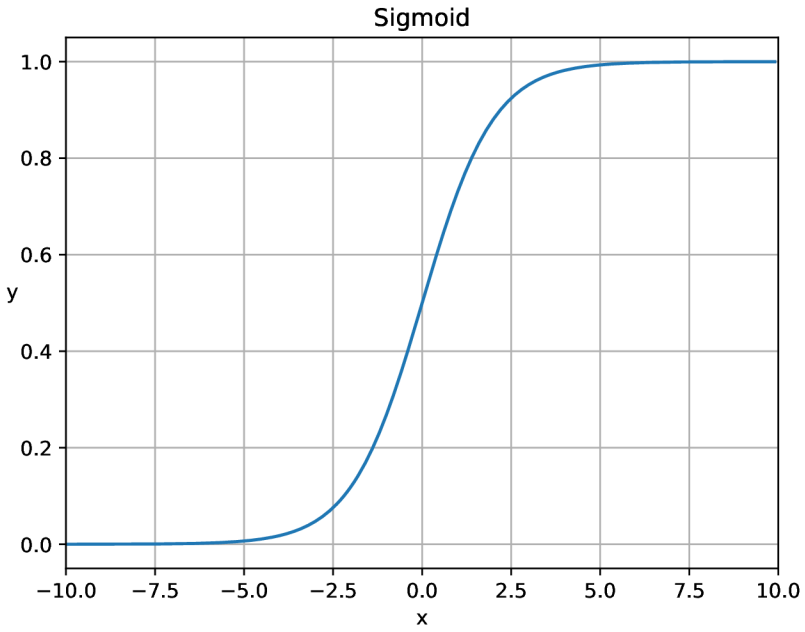
2.3.1 Aktivačné funkcie

Aktivačné funkcie prinášajú nelineárnu funkcionalitu do inak bežne lineárneho systému neurónov. Týmto umožňujú riešiť komplexnejšiu množinu úloh. Je bežné používať jeden typ aktivačnej funkcie pre všetky neuróny v neurónovej siete. Existuje veľké množstvo aktivačných funkcii a vhodne zvolená nám môže pomôcť pri riešení úlohy.

Sigmoig. Logistická funkcia sigmoida patrila historicky medzi najpopulárnejšie aktivačné funkcie, dnes sa používa iba sporadicky a skôr sa s ňou môžeme stretnúť v rámci výučných účelov. Z grafu funkcie 2.3 je zrejmé mapovanie vstupnej hodnoty do intervalu $(0, 1)$, práve to je jednou z nevýhod, lebo ako sa ukázalo neurónové siete dokážu lepšie pracovať s hodnotami centrovanými v 0. Ďalšou nevýhodou je saturácia gradientu v oblastiach blízkych 0 a 1, preto je zaradená do tzv. saturačných nelinearít [8]. Zápis funkcie vyzerá následovne

$$s(x) = \frac{1}{1 - 1 + e^{-x}}. \tag{2.2}$$

- x - potenciál neurónu



Obr. 2.3: Graf funkcie sigmoid.

Relu. V súčasnosti asi najpoužívanejšia aktivačná funkcia, v [1] ukázala niekoľko násobný nárast v rýchlosti konvergencie optimalizačnej metódy SGD 2.3.2 oproti saturačným aktivačným funkciám ako Sigmoid . Ďalšou výhodou je jej rýchlosť výpočtu, keďže ako je patrné z grafu 2.4, ovplyvňuje iba vstupné hodnoty menšie než 0, pre ktoré ponecháva výstup 0 [13, 8]. Zápis funkcie je následovný:

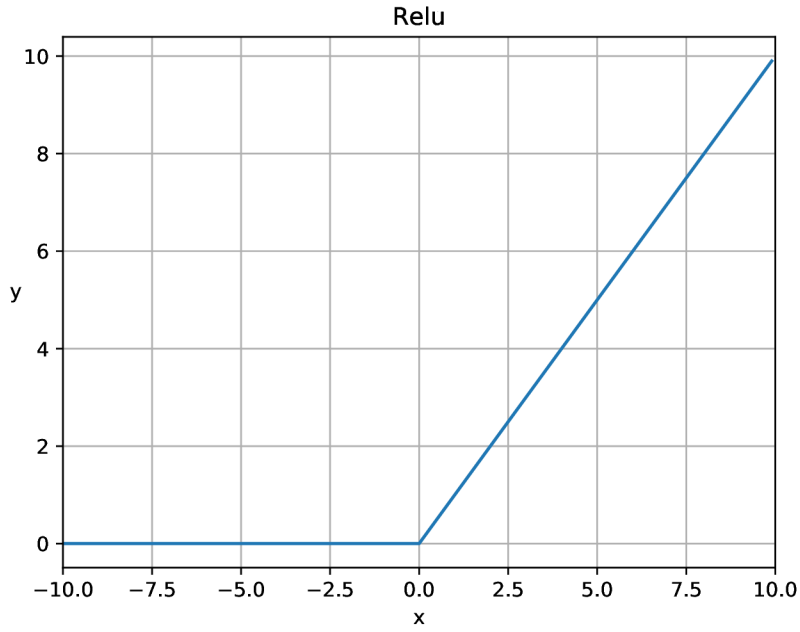
$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0. \end{cases} \tag{2.3}$$

.

- x - potenciál neurónu

LRelu. LRelu angl. Leaky Relu je modifikáciou aktivačnej funkcie Relu. Rozdiel nastáva pre vstupné hodnoty $x \leq 0$. Tie sú v bežnom Relu mapované na výstup 0. Táto skutočnosť môže mať za následok umŕtvenie neurónu pri spätnom prechode, tak že sa už nikdy neaktivuje. Preto LRelu poskytuje parameter a, ktorým sú tieto vstupné hodnoty vynásobené. Hodnota parametru a je konštanta zvolená pred tréningom modelu [8]. Funkciu preto môžeme zapísať následovne:

$$f(x) = \begin{cases} x & x \geq 0 \\ ax & x < 0. \end{cases} \tag{2.4}$$



Obr. 2.4: Graf funkcie Relu.

- x - potenciál neurónu
- a - konštanta

PRelu. Poslednou alternatívou, ktorú spomeniem je aktivačná funkcia PRelu. Narozdiel od LRelu nie je parameter a konštantný ale adaptovaní v rámci učenia siete.

2.3.2 Učenie

Tréning neurónovej siete môžeme popísať v troch krokoch:

1. Ohodnotenie správnosť výstupu voči očakávaným dátam - od toho je chybová funkcia.
2. Vypočítať závislosť jednotlivých parametrov na výstup siete - algoritmus spätného šírenia chyby.
3. Na základe zistenej závislosti optimalizovať parametre siete, aby sa zvýšila presnosť siete - optimalizácia založená na gradiente.

Chybová funkcie

V praxi sa môžeme stretnúť aj s pojmom objektívna funkcia. Slúži k ohodnoteniu presnosti trénovaného modelu neurónovej siete, porovnaním výstupu voči reálnym dáta, ktoré boli očakávané na výstupe. Čím je funkčná hodnota chybovej funkcie alias chyba nižšia tým menší rozdiel medzi predikovanými a očakávanými dátami. Hodnota chyby je ale ovplyvnená aj typom zvolenej chybovej funkcie. Existuje veľké množstvo chybových funkcií, vhodný výber závisí hlavne na úlohe, ktorú sa model snaží vyriešiť. Tu popisujem, konkrétne dve chybové funkcie použité v rámci tejto bakalárskej práce.

Cross entropy

Pokiaľ neurónová sie rieši klasifikačný problém, často sa môžeme stretnúť s kombináciou funkcie softmax a chybovej funkcie cross entropy.

Softmax - k vysvetleniu tejto funkcie si uveďme príklad, majme sieť ktorej úlohou je klasifikovať vstup do jednej z N tried, na výstupu teda dostaneme vektor reálnych čísel o veľkosti N. Pokiaľ položíme tento vektor na vstup funkcii softmax, získame pravdepodobnostnú distribúciu nad všetkými triedami so sumou v 1. Pre zápis funkcie viď 2.5, kde x_i je í-ta hodnota z výstupného vektoru siete [3, 7].

$$S(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{2.5}$$

Cross entropy použijeme k porovnaniu distribúcie získanej z funkcie softmax a reálnej distribúcie. Keďže vieme aká je skutočná trieda vstupných dát, reálna distribúcia bude mať na mieste skutočnej triedy 1 a na všetkých ostatných 0. Zápis funkcie 2.6, kde r je pravdivostná distribúcia a p predikovaná distribúcia [7].

$$H(r, p) = -\sum_x r(x) log(p(x)) \tag{2.6}$$

Stredná kvadratická chyba

Stredná kvadratická chyba angl. mean squared error alebo MSE. Povedzme, že model neurónovej siete mapuje vstupné reálne dáta x na výstupné y. MSE spočíta euklidovskú vzdialenosť medzi predikovanými hodnotami $\hat{y}$ a pravdivostnými hodnotami y. Výsledok chyby je ešte vydelený počtom vstupných hodnôt [3]. MSE môžeme teda zapísať následovne

$$M = \frac{1}{m} \|\hat{y}_i - y_i\|_2^2. \tag{2.7}$$

Algoritmus spätného šírenia chyby

Hodnota chyby, získaná z ľubovolnej chybovej funkcie aplikovanej na výstup neurónovej siete nám hovorí ako sa model správa, respektívne ako je vzdialený od skutočných hodnôt očakávaných na výstupe. Pre optimalizáciu parametrov siete potrebujeme nadobudnúť informáciu o ich vplyve na výstup. Tú získame vypočítaním gradientu chybovej funkcie $\nabla f(x, y)$, kde x sú parametre modelu, ktorých parciálnu deriváciu chceme poznať, a y sú vstupné premenné, ktorých derivácia nás nezaujíma. Algoritmus spätného šírenia chyby postupne prechádza modelom siete v opačným smerom, od chybovej funkcie až po vstupnú vrstvu a reťazovým pravidlom počíta gradient [3].

Pre príklad si uveďme jednoduchú zloženú funkciu $z = g(f(x))$ kde $y = f(x)$ a chceme zistiť parciálnu deriváciu premennej x vzhľadom k z, pre tento prípad môžeme výpočet reťazovým pravidlom zapísať následovne

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{2.8}$$

Optimalizácia založená na gradiente

Cieľom optimalizačných algoritmov je minimalizovať hodnotu chybovej funkcie. Optimalizácie založené na gradiente využívajú gradient chybovej funkcie, vypočítaný pomocou

algoritmu spätného šírenia chyby k jej minimalizácií. Gradient je vlastne vektor parciálnych derivácii, ukazujúci smer najväčšieho rastu funkcie. Ak teda chceme nájsť minimum funkcie, musíme sa vybrať opačným smerom, teda zmeniť znamienko gradientu $-\nabla f(x)$, takto získame smer najväčšieho klesania.Iteračná metóda, ktorá postupuje v smere najväčšieho klesania sa nazýva gradientový zostup angl. gradient descent alebo GD a môžeme ju zapísať ako

$$x' = x - \varepsilon \nabla f(x) \tag{2.9}$$

kde x je stará hodnota parametru, x' je adaptovaná hodnota parametru a ε určuje veľkosť kroku akým sa algoritmus vyberieme v smere najväčšieho klesania. Veľkosť kroku, je podstatnou premenou v učení neurónových sietí, ovplyvňuje rýchlosť učenia ale aj presnosť. Príliš vysoká hodnota môže spôsobiť divergenciu a príliš nízka nemusí nájsť optimálne minimum funkcie [3].

Sochastický gradientový zostup (SGD) je úpravou tejto metódy. Namiesto výpočtu gradientu z celej dátovej sady, je vždy použitá iba jedna náhodne vybraná vzorka, týmto spôsobom urýchľuje konvergenciu.

2.4 Konvolučné neurónové siete

Bežný model neurónových sieti s plne prepojenými vrstvami nie je vhodným riešením pre spracovanie viac dimenzionálnych vstupov akými sú napríklad obrázky. Výpočet sa stáva veľmi rýchlo pamäťovo ale aj výpočtovo náročný. Pre príklad si vezmime vstupný obrázok s veľkosťou 600x600x3, v poradí výška, šírka a počet kanálov (RGB). Ak by sme chceli použiť bežnú neurónovú sieť, museli by sme vstup roztiahnuť do vektora o veľkosti 1 080 000x1 tzn. pre každý neurón v prvej skrytej vrstve by sieť musela uchovávať 1 080 000 váh. Ďalšou nevýhodou, je že neurón pracuje vždy iba s jednou vstupnou hodnotou, v tomto prípade pixelom, a teda nemá poňatie o jeho okolí. Preto sa používajú konvolučné neurónové siete, ktoré k tomuto problému pristupujú inak [6].

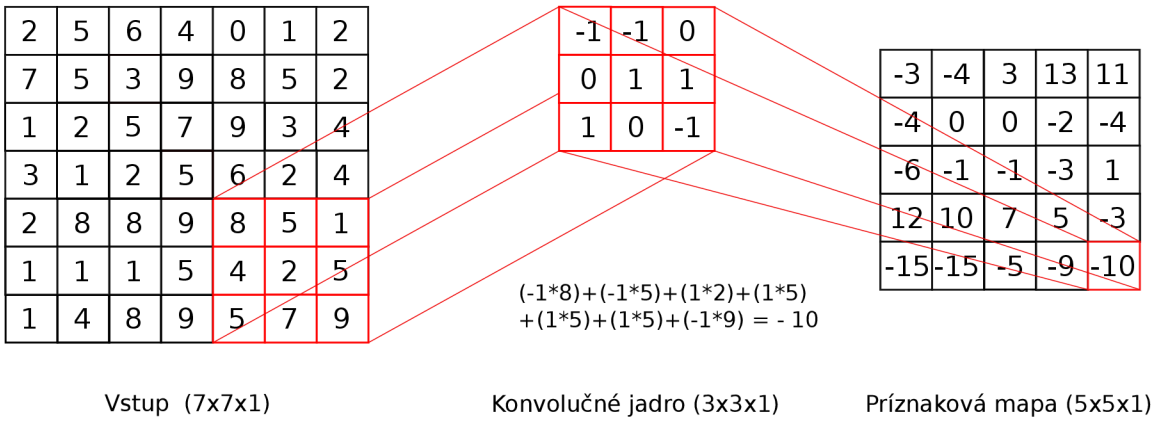
Konvolučná vrstva

Konvolučná vrstva je špeciálna vrstva používaná v neurónových sieťach, namiesto bežného neurónu v nej nájdeme konvolučné jadrá. Tie si môžeme predstaviť ako mriežky s predom stanovenou veľkosťou, kde v každom okne mriežky sa nachádza váhový faktor. Práve hodnota týchto váh je pri učení adaptovaná. Konvolučné jadro funguje na princípe posuvného okna, po vstupe sa posúva v smere zľava doprava, zhora dole. Pri každom posune je spočítaný skalárny súčin medzi váhami v jadre a hodnotami na mieste kde toto jadro prilieha. Treba si povšimnúť, že výpočet je vykonaný nad celou hĺbkou vstupu, preto je hĺbka jadra totožná s hĺbkou vstupu. Je to logické, ak uvážime vstupný obrázok s tromi farebnými kanálmi (červená, zelená, modrá), tak spracovaním informácii zo všetkých troch kanálov získame omnoho viac informácii ako spracovaním jedného. Môžeme teda povedať, že jadro pri každom posune mapuje VxSxC hodnôt zo vstupu na jednu hodnotu na výstupe, kde V je výška jadra, S je šírka jadra a C je hĺbka vstupu. Výstup konvolučnej vrstvy môžeme nazvať príznakovou mapou, jej hĺbka sa rovná počtu konvolučných jadier vo vrstve, ktorá ju vygenerovala. Princíp spracovania vstupu jedným konvolučným jadrom je zobrazený na obrázku 2.5.

Zvolenie vhodnej architektúry je kapitola sama o sebe. Rôzne modeli pracujú s rôzne veľkými jadrami. Prvá konvolučná vrstva zväčša vykonáva detekciu jednoduchých údajov

z vstupného obrázku akými sú napríklad okraje. S pribúdajúcimi vrstvami sa komplexnosť detekcie zvyšuje. Ako aj u bežných sietí je potrebné nájsť správny kompromis medzi výkonom a presnosťou. V prípade konvolučných vrstiev je nutné nastaviť tieto parametre:

- počet konvolučných jadier - viac jadier, väčšia hĺbka mapy príznakov viac informácii pre ďalšiu konvolučnú vrstvu
- veľkosť konvolučných jadier - najviac sú používané veľkosti 1x1 3x3 7x7
- krok - o koľko hodnôt je jadro vždy posunuté
- výplň nulami - na vonkajší okraj vstupného obrázku pridá vrstvu núl



Obr. 2.5: Princíp výpočtu príznakovej mapy konvolučným jadrom zo vstupu.

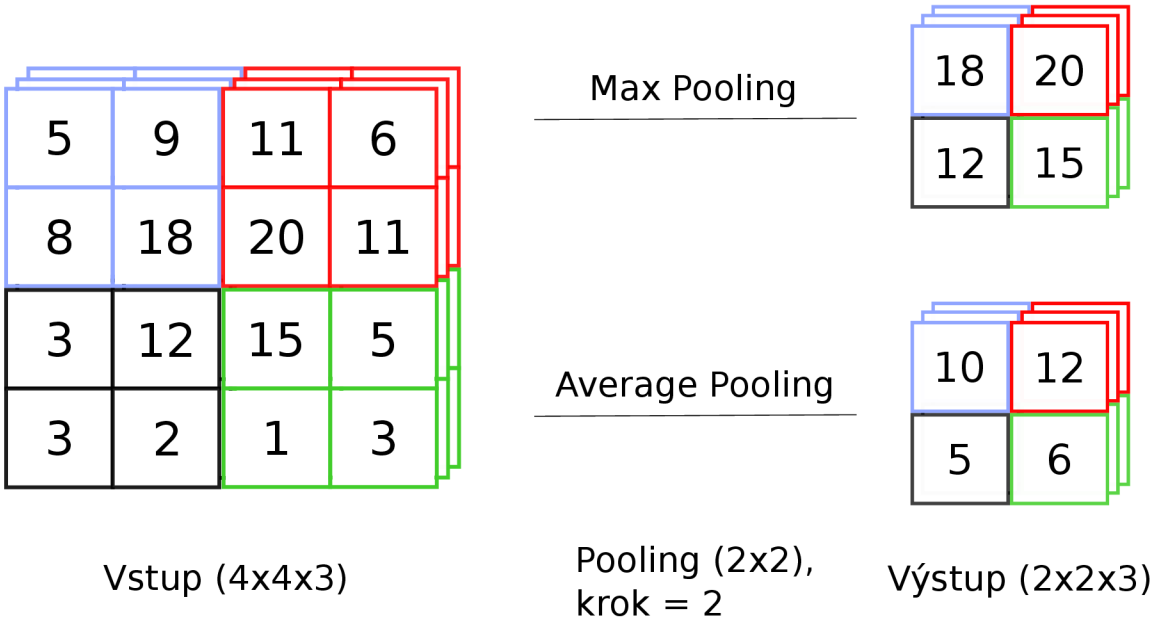
Aj v konvolučných sieťach sa stretáme s plne prepojenými vrstvami, ktoré sú väčšinou umiestnené pred výstupnou vrstvou. Aby sme mohli prepojiť konvolučnú vrstvu s plne prepojenou, musíme mapu príznakov roztiahnuť do jedného vektora. Napríklad pre výstup konvolučnej vrstvy s rozmermi 20x20x10 vynásobením získame rozmer vektoru, teda 4000x1.

Poolingová vrstva

Na mapu príznakov je aplikovaná aktivačná funkcia. Po ktorej môže nasledovať ďalšia špecifická vrstva využívaná v konvolučných sieťach a tou je vrstva poolingová. Používa sa k zmenšovaniu rozmerov príznakovej mapy, s účelom zníženia výpočtovej a pamäťovej náročnosti. Narozdiel od konvolučnej vrstvy pracuje nezávislo na každej vrstve z mapy príznakov zvlášť, teda ovplyvňuje šírku a výšku ale nie hĺbku. Jadro sa posúva po vstupe rovnako ako konvolučné jadro, zľava doprava zhora dole a nad každou oblasťou vykoná poolingovú operáciu.

K znižovaniu rozmerov sa používajú dva typy operácii, max pooling - z príslušnej oblasti vyberie pixel s najvyššou hodnotou, average pooling - priemer z pixelov v príslušnej oblasti viď obrázok 2.6. V dnešnej dobe sa však stretávame hlavne s verziou max poolingu. U poolingovej vrstvy nastavujeme tieto parametre:

- veľkosť poolingového jadra - väčšinou sú použité rozmery 2x2, 3x3
- krok o koľko hodnôt má byť poolingové jadro posunuté



Obr. 2.6: Poolingové operácie. Porovnanie max poolingu a average poolingu pri použití poolingového jadra s veľkosťou 2x2 a krokom 2.

Použitie konvolučných sietí

Konvolučné neurónové siete našli využitie v mnohých oblastiach počítačového videnia, napríklad klasifikácia, detekcia, segmentácia.

Kapitola 3

Výber modelu a množiny objektov

V tejto kapitole najskôr rozoberám ako som sa rozhodol pre výber modelu MTCNN [15] a to v krátkej sekcií 3.1. Následne sa celý zostatok kapitoly 3.2 zaoberá, popisom a princípom fungovania tohto modelu.

3.1 Výber modelu

Náplňou mojej bakalárskej práce je implementácia a následné experimentovanie nad existujúcim modelom konvolučných neurónových sietí, použiteľným k detekcii objektov z obrázku. S vedúcim bakalárskej práce sme sme prišli k záveru, že by bolo zaujímavou témou upravenie dobre známeho kaskádového modelu MTCNN [15] používaného na detekciu tvári. Avšak s tým, že množina objektov určených k detekcií sa skladá výhradne z dopravných značiek.

3.2 MTCNN

MTCNN - (Multi-task Cascaded Convolutional Networks) model pre detekciu tváre z obrázku. V kaskádovom prepojení spája tri konvolučné neurónové siete a tie riešia trojicu úloh:

- Binárna klasifikácia - (je tvár/nie je tvár)
- Regresia ohraničujúcich boxov
- Nájdenie landmarkov

S MTCNN prišla v roku 2016 skupina čínskych vedcov. Tento vydarený model dokázal v presnosti poraziť všetky state-of-art detektory tvári pri zachovaní real-time výkonu, aby som bol presný, na grafickej karte NVIDIA TITAN BLACK dokázal spracovať 99 snímkov za sekundu. Koeficient presnosti modelu sa pohybuje okolo 95 percent. Model pozostáva z troch sieti P-Net, R-Net a O-Net prepojených v poradí za sebou. Fungovanie modelu je pomerne komplikované, medzi jednotlivými sieťami, sú použité ďalšie algoritmy, takže zdrojový kód je obsiahlejší ako u bežných sietí [15].

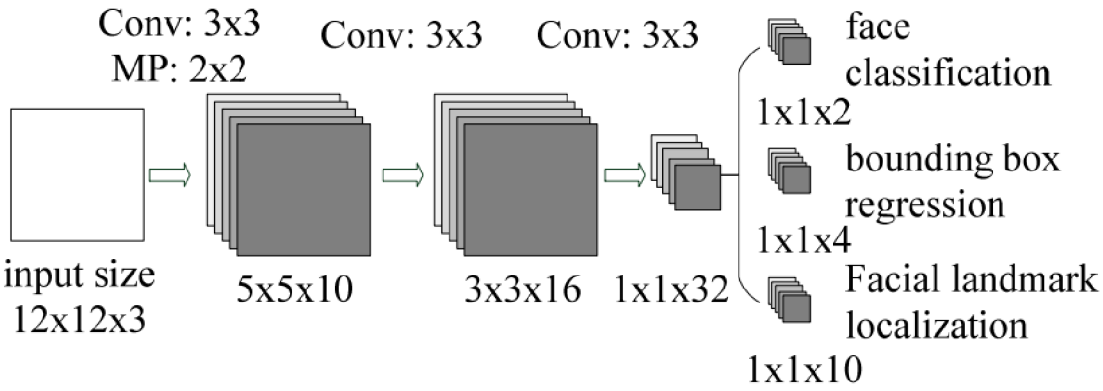
Sieť P-Net

P-Net je prvou a zároveň najmenšou sieťou celého modelu, architektúra je zobrazená na obrázku 3.1. Súčasťou siete sú štyri konvolučné a jedna poolingová vrstva. Zaujímavosťou je

absencia plne prepojenej vrstvy na konci modelu, o takejto sieti hovoríme, že je plne konvolučná. Chýbajúca plne prepojená vrstva má svoj význam, tento návrh umožňuje spracovanie vstupu s ľubovolnou veľkosťou. Zo siete získame tri výstupy:

- 2 hodnoty pre binárnu klasifikáciu
- 4 hodnoty pre regresiu ohraničujúcich boxov x1, y1 ,x2, y2
- 10 hodnôt pre pozíciu landmarkov

Veľkosť siete je dôležitým faktorom, hlavne v rýchlosti spracovania vstupných obrázkov. V pôvodnom článku o MTCNN uviedol autor vo svojej práci experiment, ktorý hovorí o enormnej rýchlosti, konkrétne 300 dopredných prechodov sieťou za 0.031s na grafickej karte. Toto je dôležitým faktorom s ohľadom na úlohou siete, zamerať podozrivé regióny vo vstupnom obrázku. Vo všetkých troch sieťach je použitá aktivačná funkcia PRelu.



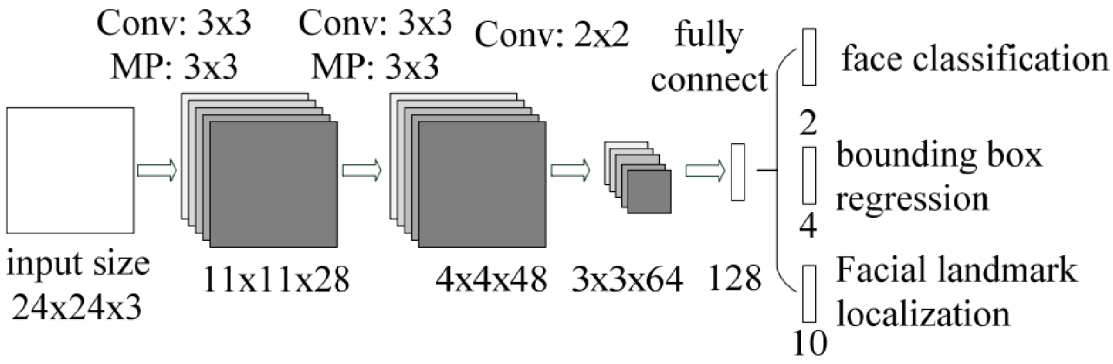
Obr. 3.1: Architektúra sieťe P-Net. Obrázok prevzatý z [15].

Sieť R-Net

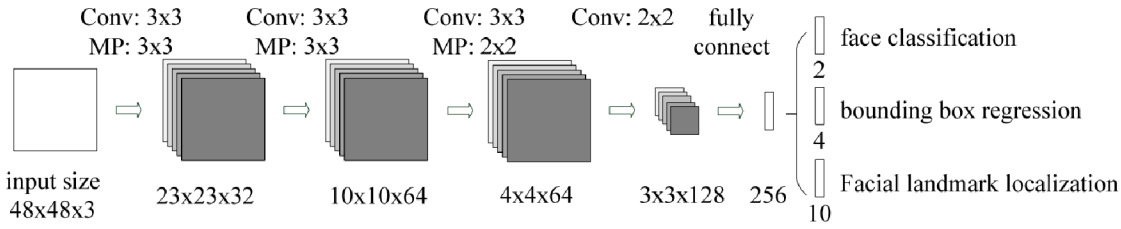
Architektúra druhej siete je veľmi podobná tej prvej, odlišuje sa poslednou vrstvou, kde bola konvolučná vrstva nahradená dvomi plne prepojenými vrstvami viď 3.2. Taktiež sa zväčšil rozmer vstupu a to dvojnásobne na 24x24 pixelov. Výstup zostal rovnaký pre všetky 3 úlohy, avšak pozorný čitateľ by si mohol všimnúť zmenu vo formáte výstupu, to je spôsobené práve výmenou konvolučnej za plne prepojenú vrstvu. Účelom tejto sieti, je zredukovanie obrovského množstva falošne pozitívnych regiónov, vygenerovaných sieťou P-Net. Väčšie rozmery vstupu poskytujú lepšie informácie o pozícii tváre v pozorovanom regióne a teda lepšiu kalibrácia ohraničujúcich boxov.

Sieť O-Net

Posledná sieť je zároveň aj najväčšou. V porovnaní so sieťou R-Net obsahuje jednu konvolučnú vrstvu naviac. Taktiež rozmery sa opäť zdvojnásobili, na 48x48 pixelov viď 3.3 Tieto malé zmeny v architektúre siete sú dobre odôvodnené, keďže výstup O-Net je výstupom celého modelu, musí poskytnúť sieti čo najviac informácii o podozrivých regiónoch k zvýšeniu presnosti ale zároveň zamedziť degradácii rýchlosti kaskádového modelu, prípadným nadmerným navýšením parametrov siete.



Obr. 3.2: Architektúra sieťe R-Net. Obrázok prevzatý z [15].



Obr. 3.3: Architektúra sieťe O-Net. Obrázok prevzatý z [15].

3.2.1 Princíp detekcie MTCNN

Princíp detekcie sa dá rozdeliť do 3 fáz, kde každá ďalšia je závislá práve na výstupe tej prechádzajúcej:

Prvá fáza

Zo vstupného obrázka je ešte pred započatím detekcie škáľovaním vytvorená tzv. pyramída obrázkov pozostávajúca zo zmenšených kópií originálu. Každá z týchto kópií je poslaná na vstup prvej sieti P-Net. Veľkosť výstupu je pritom priamo závislý na veľkosti vstupného obrázku, to vychádza zo skutočnosti, že P-Net je plne konvolučná sieť, ktorá mapuje jednu výstupnú hodnotu pre každý región s veľkosťou 12x12 pixelov. To taktiež znamená, že model dokáže úspešne odhaliť iba tváre s maximálnym rozlíšením 12x12 alebo pokiaľ sú tejto hodnote relatívne blízko. Pri reálnych dátach sa však model s takouto situáciou stretne skôr sporadicky a práve preto pracuje so zmenšenými kópiami vstupného obrázkov, pretože i tvár s pôvodnou veľkosťou 50x50 je na jednej kópii eventuálne zmenšená do oblasti 12x12. P-Net vo svojich prechodoch vygeneruje enormné množstvo ohraničujúcich boxov, z tých sú ponechané iba tie o ktorých si je aspoň na 60 percent istá, že sa v nich nachádza tvár. V ďalšom kroku sú eliminované boxy s takmer duplicitnou pozíciou pomocou algoritmu nms 3.2.1. Posledným krokov v prvej fázy je kalibrácia ohraničujúcich boxov na základe výstupu regresie.

Druhá fáza

Model vezme boxy označené za podozrivé v prvej etape a na základe nich vytvorí výrezy z originálneho obrázku. Výrezy sú poslané na vstup druhej siete v poradí, R-Net. Tá už

negeneruje žiadne nové boxy, ale sústreďuje sa hlavne na presnejšie určenie pozície. Jej účelom je spresniť staré a zbaviť sa čo najväčšieho množstva falošne pozitívnych. Podobne ako u siete P-Net, zahadzuje boxy s klasifikačným skóre nepresahujúcim hodnotu 0.7. A následný postup je rovnaký ako v prvej fáze.

Tretia fáza

Začiatok je totožný s druhou fázou. Výrezmi je nakŕmená najväčšia sieť O-Net, z jej výstupu ponecháme iba boxy s klasifikačným skóre 0.7 a väčším. Na rozdiel od predchádzajúcich dvoch etáp je vymenené poradie, najskôr model aplikuje kalibráciu boxov a až potom nasleduje algoritmus nms. Posledným krokom v celom modeli, je výpočet landmarkov pre boxy, ktoré zostali.

Výpočet chyby

Pre každú úlohu, ktorú MTCNN rieši je použitý výpočet samostatnej chybovej funkcie. Klasifikačnú časť ohodnocuje chybová funkcia cross entropy 2.6, presnosť regresie ohraničujúcich boxov a pozície landmarkov sú ohodnotené strednou kvadratickou chybou 2.7. Hodnoty všetkých troch chýb sú následne vynásobené prednastavenými váhami a sčítané z čoho získame konečnú chybovú hodnotu použitú pre algoritmus spätného šírenia chyby. Tento výpočet je možné znázorniť zápisom 3.1, kde $det, box, landmark$ predstavujú jednotlivé chyby v poradí klasifikačná, presnosť ohraničujúcich boxov a presnosť landmarkov, s príslušnými váhami α, β, L.

$$min \sum_{i=1}^{N} \sum_{j \in \{det,box,landmark\}} \alpha_j \beta_i^j L_i^j \tag{3.1}$$

Regresia ohraničujúcich boxov

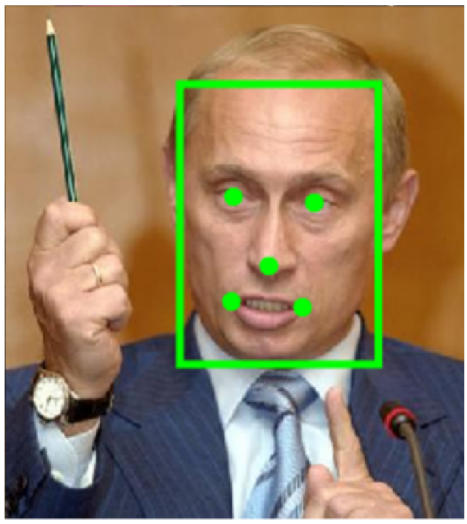
Ako bolo spomenuté už vyššie, MTCNN generuje na výstupe štyri hodnoty x1, y1, x2, y2 pre každý jeden ohraničujúci box. Dvojica x1, y1 vyjadruje posun ľavého horného rohu boxu, x2 a y2 naopak pravého dolného rohu, tieto hodnoty sú ale normalizované do intervalu $< -1, 1 >$. Takže pri kalibrácii musíme najskôr vynásobiť x1, x2 šírkou boxu a hodnoty y1, y2 výškou boxu. Získané hodnoty reprezentujú skutočný posun, teda pričítaním k pôvodným pozíciám získame kalibrovaný box.

Landmarky

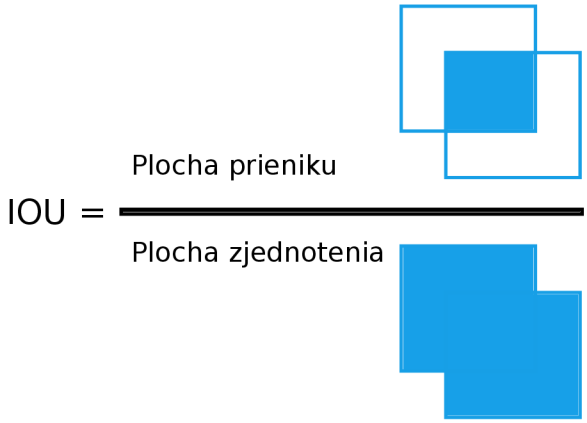
MTCNN v rámci svojej činnosti, okrem detekcie určuje päť landmarkov (orientačných bodov). Jeden bod pre každé oko, jeden pre nos a zostávajúce dva pre kútiky úst. Ako to vyzerá v skutočnosti je zobrazené na obrázku 3.4.

IOU metrika

Metrika IOU alias Intersection over Union, sa používa zväčša na ohodnocovanie správnosti predpovedaný boxov z výstupu detektorov. Princíp je veľmi jednoduchý, stačí vydeliť plochu na ktorej sa pretínajú dva boxy, plochou ich zjednotenia, viď obrázok 3.5. Výsledkom je hodnota spadajúca do intervalu $< 0, 1 >$. Čím je väčšia plocha prieniku tým je hodnota vyššia. Je základom momentálne najpoužívanejšej metriky v oblasti detekcie mAP (mean average precision). Za všeobecne dobrý odhad sa považuje, hodnota IOU väčšia ako 0.5, ale napríklad MTCNN má hranicu až na 0.65.



Obr. 3.4: Detekovaná tvár spolu s landmarkami. Obrázok prevzatý z [14].



Obr. 3.5: Výpočet metriky IOU na základe dvoch prekrývajúcich sa boxov. Obrázok prevzatý z [10].

Algoritmus NMS (non maximum suppresion)

Je možné dohľadať viacero implementácii, ktoré sa od tu spomenutej nejakým spôsobom líšia, ja popisujem konkrétne použitú v rámci modelu MTCNN. NMS algoritmus používa vyššie spomenutú metriku IOU, s cieľom eliminácie takých boxov, ktorých hodnota IOU s akýmkoľvek iným boxom presahuje hraničnú hodnotu. Rozhodnutie o tom ktorý zo vzájomne sa prekrývajúcich boxov ma byť odstránený je závislá na klasifikačnom skóre konkrétneho boxu. Algoritmus postupuje v 5 krokoch:

1. boxy sú zoradené zostupne od toho s najvyšším skóre po najmenšie
2. vyberieme box s najvyšším skóre, vypočítam jeho IOU so všetkými ostatnými
3. zmaž z pola tie boxy s IOU $>$ hranica
4. vyber ďalší box v poradí a pokračuj v 3. kroku
5. pokiaľ nie je ďalší box k výberu, ukonči algoritmus

Pri výstupe tretej siete O-Net je použitá iná variácia nms algoritmu, pretože tento výstup je zároveň aj výstupom celého modelu. Aby sme nemali vo výsledku duplicitné boxy potrebujeme zaistiť, že pre každú úspešnú detekciu značky zostane iba jeden ohraničujúci box. To dosiahneme tým, že výpočet IOU nahradíme výpočtom plochy zjednotenia boxov voči celkovej ploche jedného z týchto boxov, inak postup algoritmu zostáva nezmenený.

Kapitola 4

Dátové sady

V tejto kapitole sa zaoberám výberom vhodných dátových sád pre účeli trénovania MTCNN modelu na dopravných značkách. Najskôr popisujem sadu GTSDB v sekcii 4.1 a následne GTSRB v sekcii 4.2, to sú dátové sady použité v rámci tejto bakalárskej práce. V poslednej časti tejto kapitoly 4.3 znázorňujem generovanie nových dátových sád z týchto dvoch, tak aby boli použiteľných na trénovanie modelu MTCNN.

4.1 German Traffic Sign Detection Benchmark

Dátová sada väčšinou označovaná pod skratkou GTSDB[4] bola vytvorená skupinou výskumníkov na inštitúte neuroinformatiky v nemeckom Bochume pre účeli sútaže IJCNN-2013 Traffic sign detection benchmark. Celkovo obsahuje 900 obrázkov viď 4.2 vo formáte .ppm, vyhotovených v reálnej premávke. Rozdelené sú do dvoch skupín, sada určená k tréningu pozostáva z 600 obrázkov a 300 ostalo pre testovaciu. Nie na všetkých sa ale nachádza značka, konkrétne v trénovacie sade ja takýchto obrázkov 94 a v testovacej 63.

Dátová sada obsahuje anotáciu 43 tried značiek, ktoré čitateľ môže nájsť na obrázku 4.1. Všetky anotované značky sú vyrezané a uložené do zložiek podľa triedy do ktorej patria. Pretože väčšina obrázkov obsahuje iba jednu značku a niektoré dokonca žiadnu, množina dát je pomerne malá. V rámci sady určenej k tréningu sa nachádza 852 jednotlivých značiek. Pre testovaciu sadu je potom toto číslo ešte nižšie a to konkrétne 361.

4.2 German Traffic Sign Recognition Benchmark

Nedostatok dát k tréningu v dátovej sade GTSDB ma donútil k hľadaniu alternatívy s väčším objemom dát. Natrafil som na sadu GTSRB[12] pôvodne určenú ku klasifikácii, ktorá bola zhodou náhod vytvorená tou istou skupinou výskumníkov iba o dva roky skorej. Ako som už spomenul je určená ku klasifikácii a preto neobsahuje celé obrázky ale iba výrezy značiek s kúskom pozadia. Obrovskou výhodou je veľký objem dát, 39209 výrezov pre tréning a 12630 testovacích výrezov. Celkovo sa teda dostávame na číslo 51839, to je 50 násobný nárast oproti jeho detekčnej alternatíve. Štruktúra je rovnaká ako u prvej dátovej sady takže práca s oboma dvomi sadami sa dá veľmi ľahko skĺbiť. Značky sú rozdelené do 43 rovnakých tried. Zaujímavosťou, ktorá nebýva bežná u klasifikačných dátových sád, je anotácia značky vo výreze. Bez tohoto by som túto sadu nedokázal využiť.

Obr. 4.1: Triedy anotované v datasetoch GTSRB a GTSDB. Prevzaté z [9].

Skratky GTSBD a GTSRB sú si veľmi podobné a preto by neskôr v texte mohli čitateľa zmiasť. Z tohto dôvodu používam v ďalších častiach tejto práce prezývky. Odteraz sa budem o dátovej sada GTSDB vyjadrovať ako o detekčnej sade a o GTSRB ako o klasifikačnej sade.

4.3 Generovanie trénovacích dát

Dáta z klasifikačnej ani detekčnej dátovej sady nespĺňali všetky požiadavky, ktoré bolo potrebné splniť k tréningu sietí modelu MTCNN. K splneniu týchto požiadaviek som vytvoril skript `process_dataset.py` určený k spracovaniu a generovaniu validných dátových sád. Skript dokáže na základe vstupných parametrov programu, vygenerovať vhodné dáta pre ktorúkoľvek zo sietí P-Net, R-Net, O-Net a to nezávislo na výbere dátovej sady(detekčnej, klasifikačnej). Z množiny dát je nutné vytvoriť tri skupiny vzoriek:

- pozitívne - výrezy so značkou, ktorých IOU s GT[1] presahuje hodnotu 0.65
- čiastočné - výrezy so značkou, ktorých IOU s GT je v rozmedzí 0.40 až 0.65
- negatívne - náhodné výrezy, v prípade zachytenia značky nepresiahnu hodnotu 0.3 IOU s GT

[1]Ground truth - skutočná pozícia značky

Obr. 4.2: Ukázka datasetu GTSDB.

Obr. 4.3: Ukázka datasetu GTSRB. Vzorky obrázkov pre 4 triedy značiek, postupne od vrchu trieda 1, 14, 25 a 40.

4.3.1 Pozitívne a čiastočné výrezy

Skript obsahuje obaľovaciu funkciu pre obidva základné dátové sady, takže po zvolení parametrami, načíta príslušný sú príslušný súbor s anotáciami značiek a pre každý GT box vykoná nasledujúci postup zobrazený na obrázku 4.4.

Obr. 4.4: Generovanie vzorkov do datasetu.

;

1. náhodne zmeň veľkosť boxu, hraničná hodnota je 1/5 veľkosti GT ohraničujúceho boxu
2. vygeneruj náhodný posun na ose x a nezávisle na ňom posun na ose y
3. vyrež časť, ktorú ohraničuje nový box, a na základe IOU hodnoty medzi novým a GT boxom ulož do novej dátovej sady ako jedno z (pozitívne, negatívne čiastočné)

Algoritmus generovanie je možné nájsť vo funkcii `createRandomly`. Do anotačného súbora sú o každom výreze ukladané nasledujúce údaje: meno obrázka, štyri desatinné hodnoty

značia posun výrezu voči GT boxu, tie používame pri trénovaní regresie ohraničujúcich boxov, ukladáme ich v nasledujúcom poradí `[posun ľavého horného rohu na ose x, posun ľavého horného rohu na ose y, posun pravého dolného rohu na ose x, posun pravého dolného rohu na ose y]`, hodnoty posunu jednoducho získame odčítaním nového boxu od originálneho a pred uložením ich normalizujeme podľa veľkosti obrázku v akej ich ukladáme, poslednou hodnotou, ktorú ukladáme je index triedy do ktorej spadá konkrétna značka z obrázku.

4.3.2 Negatívne výrezy

Autori Mtcnn vo svojom článku spomínajú pomer dát pri tréningu 3:1:1 a to v poradí negatívne, pozitívne, čiastočné. To znamená, že ak chcem trénovať siete na dátach vygenerovaných z klasifikačnej dátovej sady, čo v prípade využitia všetkých dát dáva 39 209 pozitívnych/čiastočných vzoriek, musím vygenerovať aspoň 120 000 negatívnych. V skutočnosti je toto číslo ešte väčšie keď vezmeme do úvahy veľkosť batchu, ktorý nieje vždy možné presne zložiť v takomto pomere. Preto som pre účeli trénovania generoval až 140 000 negatívnych výrezov. Ich generovanie bolo komplikovanejšie, už iba z toho dôvodu, že klasifikačná dátová sada pozostáva výhradne z výrezov značiek, tým pádom jedinou možnosťou zostalo ich generovanie z detekčnej dátovej sady. Takto vznikala kombinácia medzi týmito dátovými sadami.

Algoritmus generovania negatívnych vzorkou je celkom triviálny, výrezy generujem iba z obrázkov, na ktorých sa nenachádza ani jedna značka. Takto urýchlim generovanie pretože nemusím kontrolovať pozíciu vyrezaného regiónu voči GT boxom. Ako som už spomínal pri popise, obrázkov bez akejkoľvek značky je v sade detekčnej dátovej sady len 94, ak si týmto číslom vydelíme počet výrezov, ktorý chceme generovať zistíme, že z každého obrázku musí vzísť takmer 1500 výrezov. Veľkosť i pozíciu boxov generujem čisto náhodne.

4.3.3 Generovanie z výstupu siete

Implementoval som aj ďalšiu možnosť generovania dátovej sady, a to z výstupu ľubovolnej fázy modelu. V praxi to znamená, že oblasti v obrázku vyhodnotené sieťou ako značky, ktoré by boli inak poslané na vstup ďalšej siete sú uložené ako súčasť novej dátovej sady. Z praktického hľadiska mám záujem iba o výstup prvej a druhej fázy, pretože tretia je zároveň výstupom celého modelu. S týmto účelom je implementovaná funkcia `create_dataset` v module `run_it.py` tá berie ako prvý parameter cestu k dátovej sade, na ktorej pobeží detekcia a teda zdroj generovania dát, druhý parameter špecifikuje cestu kam ukladať vygenerované vzorky. Posledný parameter určuje cieľovú fázu modelu MTCNN. Odtiaľto je volaná metóda detektoru `extract_output_sample`, starajúcu sa o celý proces generovania. Mimo iné taktiež ponúka redukciu počtu falošne pozitívnych vzoriek. Táto funkcionalita sa hodí hlavne v prípade výstupu prvej fáze, kde sa počet falošne pozitívnych pohybuje v rádoch desaťtisícov až státisícov. Parametrom `neg_delete` sa určuje koľko percent negatívnych vzoriek bude zahodených.

4.3.4 Rozdelenie dátových sád

Dátové sady použité pri trénovaní sietí sú vždy rozdelené na 2 časti a to časť s dátami k tréningu a dátami k validácii. Na tomto usporiadaní sa zakladá celý proces trénovania, všetky použité triedy a moduly. Testovacie dáta v tomto prípade niesu potrebné, predsa len jednotlivé siete trénujeme na výrezoch, ktoré nám o výslednom správaní detektora veľa

nepovedia. Z toho dôvodu je pre účeli generovania dát určených k validácii z klasifikačného dátovej sade, použitá celá testovacia časť (12 630 obrázkov).

Iná situácia nastáva v prípade detekčnej dátovej sady. Na nej budem vyhodnocovať model, preto som ju rozdelil na 3 časti. Rozdelenie vyzerá následovne:

- tréningové dáta - obrázky 0 až 399
- validačné dáta - obrázky 400 až 599
- testovacia časť - nezmenená oproti originálu, 600 až 900

Kapitola 5

Návrh a implementácia

V tejto kapitole sa zameriavam na popis návrhu riešenia a taktiež dôležitých implementačných detailov. Najskôr ale popisujem technológie použité k implementácii riešeného problému. Následne v časti 5.1 ukazujem návrh zmien týkajúcich sa jednotlivých sietí modelu MTCNN. V poslednej časti tejto kapitoly 5.2 popisujem implementačné detaily určitých aspektov projektu.

Verzovací systém

V dnešnej dobe je už takmer štandardom použitie nejakého verzovacieho systému pri vývoji softvéru. Hlavne preto, že poskytujú úplnú históriu v rámci súborov, ich jednoduché zdieľanie, dostupnosť, prístup k ľubovolnej verzii, zálohu a mnoho ďalšieho. Všetky tieto vlastnosti sú veľmi cenné pri každodennom vývoji. V rámci tejto bakalárskej práce som používal webovú službu Github[1], ktorá je založená na verzovacom nástroji git a v súčasnosti sa štíti titulom, najpoužívanejšej služby v tejto oblasti[2]. Moje rozhodnute vzišlo práve z predchádzajúcich skúseností s týmto nástrojom. Všetky zdrojové kódy vytvorené v rámci tejto bakalárskej práce sú umiestnené v repozitári https://github.com/andrejPP/MTCNN-detekcia-znaciek.

Pytorch

Open-source rámec strojového učenia pre jazyky Python a C++, vyšiel z dielne facebooku v roku 2016 a odvtedy neustále naberá na popularite. Je založený na knižnici torch[3], čo je vysoko optimalizovaný nástroj pre prácu s tenzormi na grafických kartách. Podporuje technológiu cuDNN[4] a prináša možnosť rozdelenia výpočtu medzi CPU a GPU, okrem toho je veľmi jednoduché integrovanie vlastných rozšírení a upravenie základných tried k vlastnému použitiu. Jednou z hlavných domén tohto rámcu je schopnosť zaznamenávať výpočtový graf všetkých operácií nad tenzormi za behu programu. Tento graf je následne použitý k výpočtu gradientu algoritmom spätného šírenia chyby, čím je uľahčená práca vývojárov a zrýchľuje sa tak celý proces učenia.

[1] www.github.com
[2] https://hackernoon.com/top-10-version-control-systems-4d314cf7adea
[3] https://github.com/torch/torch7
[4] https://developer.nvidia.com/cudnn

Moje rozhodnutie pre Pytorch[5] ako hlavný rámec v rámci bakalárskej práce vychádzalo z predchádzajúcich skúseností s týmto nástrojom, ktoré som nadobudol v rámci seminára o neurónových sieťach.

Numpy

Numpy[6] je knižnica pre Python, poskytujúca jednoduché a rýchle rozhranie pre vedecké výpočty. Pracuje so štruktúrami n-rozmerných polí podobnými tenzorom u Pytorchu. Numpy som využil v rámci implementácie modelu MTCNN, pre spracovanie dát medzi jednotlivými sieťami.

OpenCV

OpenCV[7] (Open Source Computer Vision Library) je bohatá knižnica s viac ako 2500 algoritmami použiteľnými v oblasti počítačového videnia a strojového učenia. Knižnica je napísaná vo vysoko optimalizovanom C++, preto sú algoritmy veľmi rýchle. Okrem iného poskytuje rozhranie pre jazyk Python. Obrázky spracované touto knižnicou sú uchovávané v n-rozmerných poliach vyššie spomenutej knižnice Numpy. Práve kvôli tomu som ju zvolil pre spracovanie vstupných obrázok môjho modelu.

5.1 Zmeny v modeloch sietí

Zmeny v návrhu sietí spočívajú iba v úprave výstupnej vrstvy. V prvom rade sú zo všetkých sietí odstránené výstupy pre landmarky. Ďalšou zmenou je úprava klasifikačného výstupu siete O-Net z binárnej klasifikácie na mnoho triednu tj. 43 tried značiek + 1 trieda pre pozadie. Výstupné vrstvy sietí je možné vidieť na obrázku 5.1

Implementácie sietí sa nachádzajú v module `nets.py`. Počet tried, pre ktoré sa vykonáva klasifikácia nie je konštantná, preto je nutné typ klasifikácie špecifikovať parametrom `classification_mode` v konštruktore objektu. Na výber máme jednu z dvoch možností:

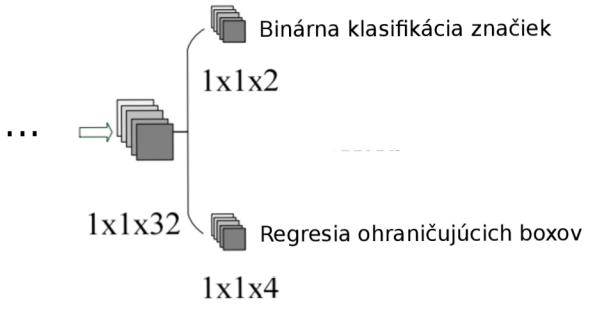
- binary : binárna klasifikácia, značka + pozadie
- multi - 44 tried, pre každú s tried značiek + pozadie

Konštruktor sietí berie ešte jeden parameter `channels`, ten upresňuje počet kanálov vstupných obrázkov. Predvolená hodnota tohto parametru je 3, počíta so vstupom farebných kanálov RGB. Toto umožňuje väčšiu voľnosť pri tréningu bez priameho zasahovania do kódu.

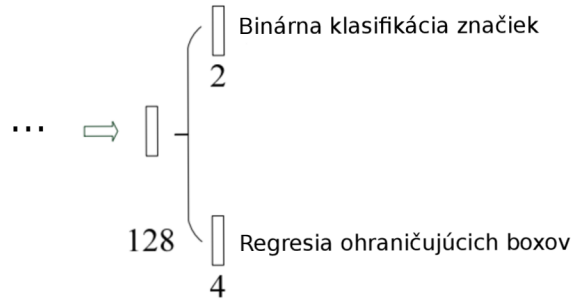
5.2 Implementácia

Tu popisujem iba ty najpodstatnejšie časti implementácie v rámci tejto práce. Bližšie informácie k ostatným zdrojovým súborom je možné nájsť na hore odkazovanom repozitári alebo na priloženom pamäťovom médiu.

[5] https://pytorch.org/
[6] https://www.numpy.org/
[7] https://opencv.org/



(a)



(b)

(c)

Obr. 5.1: Výstupné vrstvy jednotlivých sietí po úprave, (a) P-Net, (b) R-Net, (c) O-Net.

Konfigurácia trénovacích behov

S prihliadnutím na tom aby bolo možné trénovanie všetkých troch sietí bez zbytočného opakovania kódu, som vytvoril modul `training.py`. Modul dostáva na vstup ako argument cestu ku konfiguračnému súboru a prepínač –use-cuda tým zapneme počítanie na grafickej karte pokiaľ sú splnené všetky požiadavky (verzia Pytorch, grafická karta podporujúca cuDNN). Konfiguračný súbory používa zápis Json[8] a dostupné parametri sú vysvetlené v tabuľke 5.1.

Dávkovanie dát

V rámci tréningov je nutné dávkovať tri kategórie dát (negatívne, pozitívne, čiastočné) v prednastavenom pomere. Pytorch ponúka v knižnici `torch.utils.data.sampler` štyri základné typy samplerov. Naneštastie žiadny z nich nepodporuje požadované chovanie. Za tým účelom som vytvoril modul `samplers.py` s triedami `BasicSampler` a `BatchSamplerMulti`. Trieda `BatchSamplerMulti` príma v konštruktore ľubovolný počet instancii triedy `BasicSampler`, kde každý z týchto objektov je jednoduchým samplermom jednej kategórie dát. Zo všetkých vstupných samplerov generuje mini-batch v požadovanom pomere.

Spracovanie vstupného obrázku

Spracovanie vstupných obrázkov má na starosti trieda `Image_processor` z modulu `image.py`. Načíta obrázok a škálovaním vytvorí pyramídu pre účely detektoru MTCNN, k tomu využíva rýchlosti knižnice OpenCv. Okrem toho sa v module nachádzajú aj funkcie pre normalizáciu obrázkov, zmenu farebných kanálov, vykresľovanie ohraničujúcich boxov. Zmena farebných kanálov je nutná ihneď po načítaní obrázku, pretože funkcia `opencv.imread` používa poradie BGR, naproti tomu knižnica Pillow[9] na ktorej je založený Pytorch využíva poradie kanálov RGB.

Implementácia modelu MTCNN

Model MTCNN som implementoval v jazyku Python, kód sa nachádza v module `mtcnn.py`. Inšpiroval som sa originálnym kódom napísaním v jazyku matlab[10]. Detekciu je možné spustiť následovne.

```
from mtcnn import MTCNN

obrazok = "cesta_k_obrazku"
detektor = MTCNN()
pole_ohranicujucich_boxov = detektor.detect(obrazok)
```

Výstupom detekcie je n-rozmerné pole, v ktorom sú obsiahnuté všetky ohraničujúce boxy.

[8] https://www.json.org/
[9] https://pillow.readthedocs.io/en/stable/
[10] https://github.com/kpzhang93/MTCNN_face_detection_alignment/tree/master/code/codes/MTCNNv1

Parameter	Vstupná hodnota	Popis
net	[first,second, third]	typ siete, ktorá má byť trénovaná
epochs_num	celé číslo	počet iterácií nad dátovou sadou
batch_size	celé číslo	veľkosť jedného mini-batchu
model	reťazec	cesta ku konkrétnemu modelu siete vo formáte .pt, ak neexistuje je vytvorený podľa mena konfiguračného súboru
dataset	reťazec	dátová sada na ktorej pobeží tréning
class-mode	[binary, multi]	typ použitej klasifikácie
base_lr	desatinné číslo	minimálna hodnota kroku učenia
max_lr	desatinné číslo	maximálna hodnota kroku učenia
ohsm	desatinné číslo v intervale (0,1)	použije metódu ohsm, napr. 0.7 -> použi horných 70 % mini-batchu s najväčšou chybou
validate	celé číslo	ako často má byť vykonaná validácia modelu
classification_weight	desatinné číslo	hodnota, ktorou vynásobíme klasifikačnú chybu pred jej započítaním do celkovej chyby
regresion_weight	desatinné čílo	hodnota, ktorou vynásobíme chybu regresie ohraničujúcich boxov pred jej započítaní do celkovej chybovej hodnoty

Tabuľka 5.1: Popis konfiguračných parametrov a ich hodnôt.

Kapitola 6

Experimenty

V tejto sekcii popisujem priebeh trénovania sietí, aké problémy nastali a experimenty, ktoré som spracoval pri vývoji modelu. Konkrétne v sekcii 6.1 je popísaný vývoj siete P-Net, to samé pre druhú sieť R-Net môžme nájsť v sekcii 6.2. Výsledky poslednej siete O-Net sú umiestnené v sekcii 6.3. V poslednej sekcii 6.4 prezentujem výsledný výkon modelu na vyhodnocovacích dátach.

Metacentrum

Trénovanie troch sietí si kladie väčšie požiadavky na zdroje, než k akým som mal prístup. Z toho dôvodu som využil služieb Českého národného gridového centra. Spája zdroje organizácií ale hlavne univerzít po celej českej republike s cieľom počítania náročných úloh pre výskumné účeli. Ku dňu písania tejto časti, bolo v rámci metacentra dostupných 18 268 CPU [1] a 16 GPU [2]. Proces spúšťania jednotlivých úloh je spracovaný automatizovaným systémom PBS Professional, ten spustí úlohu akonáhle sú dostupné zdroje, ktoré si vyžiadala. Keďže je celý proces automatizovaný vytvoril som skripty v jazyku bash. Tie sa starajú o celý priebeh trénovania ale i o spracovanie výsledkov.

Vyhodnocovanie tréningov

S cieľom dosiahnuť najvhodnejšiu kombináciu parametrov pre čo najvyššiu presnosť siete, je potreba vyhodnocovať trénovaný model na dátach určených k validácii. Tie však pozostávajú výhradne z výrezov obrázkov, takže nie úplne presne simulujú správanie detektora na celých obrázkoch. Preto som zaradil do tréningu okrem bežnej validácie aj beh detektora na validačnej časti detekčnej sady. V prípade druhej a tretej siete, zároveň získam prehľad o tom ako dobre jednotlivé časti detektora spolupracujú.

Po vzore pôvodného modelu MTCNN, sú za pozitívne boxy považované tie, ktoré prekročia hodnotu IOU > 0.65 s akýmkoľvek pravdivostným boxom. Toto však platí len pre binárnu klasifikácii sietí P-Net a R-Net a je vyhodnocované funkciou `iou_benchmark` z modulu `benchmark.py`. U týchto sietí nám nejde ani tak o úplnú presnosť, pretože tá závisí aj od množstva falošných detekcií ale skôr o úspešnosť zachytených značiek. V angličtine sa tento spôsob vyhodnocovania nazýva recall, a môže byť zapísaný následovne

[1] https://metavo.metacentrum.cz/pbsmon2/hardware
[2] https://wiki.metacentrum.cz/wiki/GPU_stroje

$$recall = \frac{počet\ úspešne\ zachytených\ značiek}{celkový\ počet\ vygenerovaných\ boxov}. \tag{6.1}$$

U poslednej siete je nutné aby sa predpovedaná trieda priliehajúceho boxu zhodovala so skutočnou triedou daného GT boxu. Keďže zároveň predstavuje výstup celého modelu, zaujíma nás jej celková presnosť, k získaniu čoho potrebujeme poznať v akom pomere sú úspešne zachytené značky k celkovému počtu predpovedaných boxov. V angličtine je pre to názov precision a zápis vyzerá následovne

$$precision = \frac{počet\ úspešne\ zachytených\ značiek}{celkový\ počet\ predikcií} - \tag{6.2}$$

Z hodnôt Recall a Precision dokážeme vypočítať metriku f1, čo vlastne predstavuje ich váhový priemer

$$f1 = 2 * \frac{precision * recall}{precision + recall}. \tag{6.3}$$

Použité datasety

Po niekoľkých tréningových behoch na výrezoch z detekčnej dátovej sady v úvodných etapách experimentovania som prišiel k záveru, že 852 vzoriek značiek nepostačovalo k dosiahnutiu dobrých výsledkov. Z tohoto dôvodu som do trénovania zahrnul klasifikačnú dátovú sadu.

Pre každú zo sietí som vytvoril jednu základnú dátovú sadu s príslušnou veľkosťou obrázkov (12x12, 24x24, 48x48). Pozitívne a čiastočné výrezy značiek generujem z klasifikačnej dátovej sady, metódou popísanou v 4.3.1, takže ich počet sa pohybuje okolo čísla 39 200. Negatívne generujem metódou popísanou v 4.3.2 z detekčnej dátovej sady.

Krok učenia

Nevhodne zvolená veľkosť kroku môže viesť k neúspechu pri učení. Veľmi nízka hodnota spôsobuje, že trénovaný model s veľkou pravdepodobnosťou nedokáže dosiahnuť plného potenciálu, pretože sa zasekne v ne-optimálnom minime, navyše tréning trvá extrémne dlho. Naopak príliš vysoká hodnota väčšinou nedokáže nájsť optimálnu hĺbku riešenia riešenie. Dobrou taktikou je v úvodných epochách tréningu, kde sme ďaleko od minima používať vyššiu hodnotu a následne ju znižovať aby sme dokázali nájsť lepšie dno. Leslie N. Smith to vo svojej práci [11] poňala ešte trochu inak a prišla so zaujímavým konceptom adaptácie. Namiesto kontinuálne klesania je založená na cyklickom zvyšovaní a znižovaní kroku v rozmedzí predom nastavených hodnôt. To vychádza z myšlienky, že ak sa učenie zasekne v slabom lokálnom minime, zvýšením hodnoty kroku z neho dokáže uniknúť. Pokiaľ je ale robustné, tak tam zostane.

Tento koncept som využil v rámci učenia sietí. Pre nastavenie cyklu je potrebné zistiť kde sa pohybuje horná hranica kroku, ktorá ešte pridáva na presnosti. Na zistenie tejto hodnoty existuje jednoduchý test (lr range test), popísaním v spomínanej práci. Od hornej hranice sa odvíja spodná hranica v cykle, tá by mala byť v rozmedzí $\frac{1}{3}$ až $\frac{1}{4}$ hornej hranice. Takto zistené hodnoty nastavujem v konfiguračnom súbore parametrom `base_lr` pre dolnú hranicu a `max_lr` pre hornú hranicu. Posledným krokom je nastavenie dĺžky jedného cyklu, autorka odporúča hodnotu v intervale dvoch až desiatich etáp, v konfiguračnom súbore sa nastavuje parametrom `cycle`.

Všetky trénovacie behy používali optimalizačný algoritmus SGD [3].

Online hard sample mining

Metóda pri ktorej nie je chybová hodnota vypočítaná z celého mini-batchu ale iba z jeho určitej časti. Väčšinou sa používa niekoľko horných percent vzorkou s najväčšou chybou. Vychádza z faktu, že vzorky, pre ktoré sa sieť mýlila najviac dokážu lepšie upraviť a ovplyvniť parametre siete. Použitie tejto metódy je umožnené parametrom `ohsm` v konfiguračnom súbore. Jemu priradená hodnota musí byť desatinné číslo z intervalu (0,1), kde 0 - nepouži žiadne vzorky k výpočtu chyby a 1 znamená - použi všetky vzorky v mini-batchi. Vo všetkých tréningových behoch som použil hodnotu 0.7.

6.1 Tréning prvej siete P-Net

Ako prvé netreba zabudnúť, že prvú sieť trénujeme ako binárny klasifikátor a teda k tomu sa vzťahujú všetky prezentované grafy. Vždy zobrazujem iba grafy najlepšieho tréningového behu. Pre sieť P-Net sú zobrazené v 6.1.

Parametre tréningu

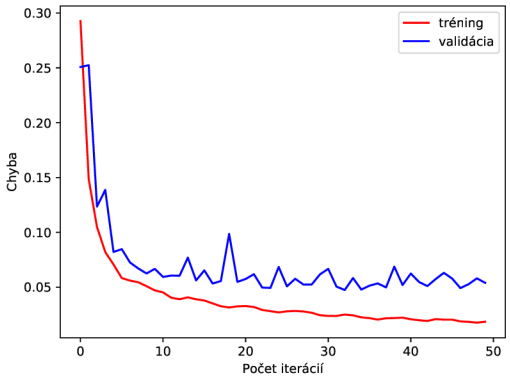
Váhy jednotlivých chýb som ponechal na hodnote 1. Testovaním som zistil, že ak je hodnota váhy, ktorou je násobená chyba regresie ohraničujúcich boxov `0.5` (použitá autormi), sieť nedosahuje svojho najlepšieho výkonu. Tréningové behy s hodnotou váhy `1` ukázali o niečo lepšie výsledky. Dôležité parametre konfiguračného súboru použité pri dosiahnutí najlepšieho výsledku.

```
{
  "net" : "first",
  "epochs_num" : 50,
  "batch_size" : 128,
  "class-mode" : "binary",
  "dataset": "data_12",
  "base_lr" : 0.005,
  "max_lr" : 0.015,
  "cycle" : 3,
  "classification_weight" : 1,
  "regression_weight" : 1
}
```

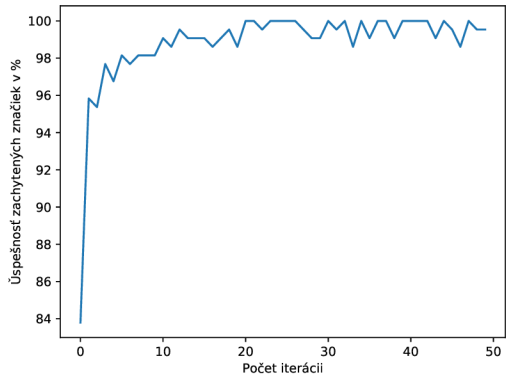
Súhrn

Graf (a) z 6.1 ukazuje, že validačná chyba prestala klesať po 20 tréningových iteráciách. To isté je zrejmé aj z grafu (b), kde sieť v tomto čase začala premenlivo zachytávať 100 % značiek. Následné skoky v presnosti sa síce môžu zdať veľké, avšak je potrebné si uvedomiť, že počet vzorkou použiteľných pre validáciu je iba 216, z čoho 2 % predstavujú po zaokrúhlení 4 vzorky. Na grafe (c) vidíme, že generuje veľké množstvo falošne pozitívnych detekcií.

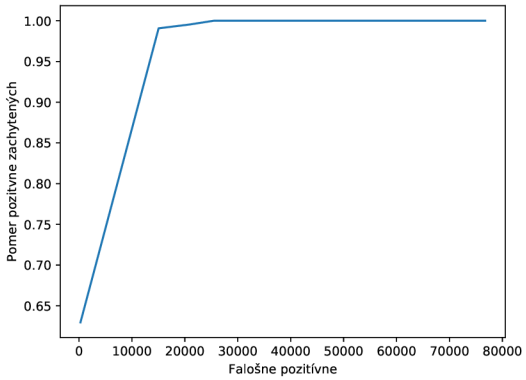
[3] https://pytorch.org/docs/stable/optim.html



(a)



(b)



(c)

Obr. 6.1: Grafy z tréningu siete P-Net. Na grafe (a) je zobrazený vývoj chyby, (b) ukazuje vývoj úspešnosti zachytených značiek a (c) znázorňuje pomer medzi zachytením značiek a falošnou detekciou.

6.2 Tréning druhej siete R-Net

Druhá sieť v poradí v rámci detekcie, taktiež vykonáva binárnu klasifikáciu. Jej účelom je eliminácia drvivej väčšiny vstupov poslaných zo siete P-Net. S tým, že nesmie degradovať počet úspešných detekcií.

Parametre tréningu

Parametre tréningu sú takmer totožné, s tými pri tréningu prvej siete. Zmena nastala iba v použitej dátovej sady a zmeneným typom siete.

```
{
  "net" : "second",
  "epochs_num" : 50,
  "batch_size" : 128,
  "class-mode" : "binary",
  "dataset": "data_24",
  "base_lr" : 0.005,
  "max_lr" : 0.015,
  "cycle" : 3,
  "classification_weight" : 1,
  "regression_weight" : 1
}
```

Experiment: porovnanie dátových sád

Z grafu (a) na 6.2 je možné vidieť vývoj úspešnosti zachytených značiek v procese trénovania a teda aj to, že k zachyteniu všetkých značiek z validačnej sady pri stávajúcich parametroch postačuje okolo 30 tréningových iterácii. R-Net dokázala zredukovať počet chybných detekcií približne 10 násobne, to je vidno na grafe (c). Po spriemerovaní to však dáva 18 falošných detekcií na obrázok, čo je stále pomerne veľké číslo. Z analýzy výstupu druhej siete som zistil, že okrem iného prepúšťa hlavne dopravné značky nezaradené do klasifikácie. Príklady takýchto chybných detekcií sú ukázané na obrázku 6.3. Tento problém je spôsobený nedostatočnou rozmanitosťou informácii obsiahnutých v negatívnej časti dátovej sady určenej k tréningu.

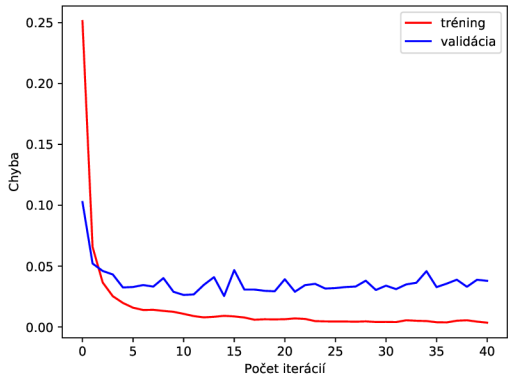
S tým cieľom porovnávam úspešnosť siete R-Net po natrénovaní na troch rôznych dátových sadách.

1. náhodne generovaná dátová sada (NH) - základná sada použitá pri predchádzajúcom tréningu siete 6, v tomto prípade jej dávam pomenovanie náhodná, lebo negatívna časť je generovaná náhodnými výrezmi z detekčnej dátovej sady a to je podstatný údaj pri tomto experimente

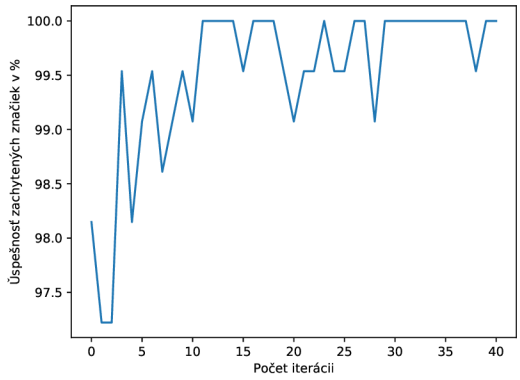
2. dátová sada generovaná z výstupu siete P-Net - popis generovania 2

3. kombinácia dátovej sady NH + z výstupu siete P-Net - kombinácia týchto dvoch tvorí najväčšiu vzorku

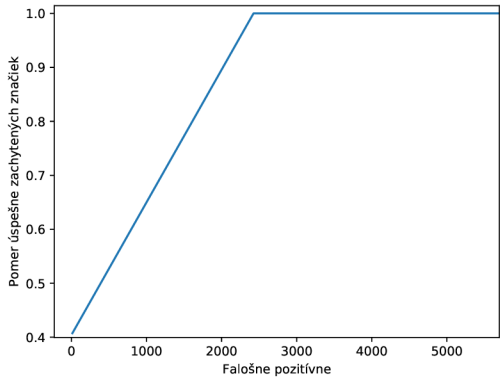
Konfigurácia parametrov tréningu je rovnaká ako pri predchádzajúcom trénovaní siete, jediné čo sa zmenilo sú použité dátové sady a v prípade dátovej sady generovanej z výstupu, je veľkosť batchu zmenšená na hodnotu 32, pretože výstup pozitívnych a čiastočných dát je o dosť menší.



(a)



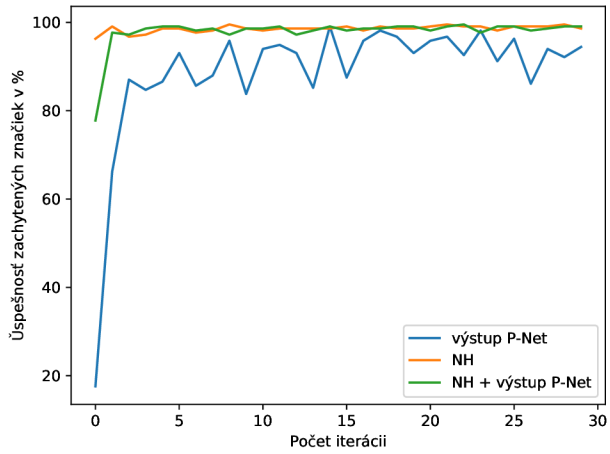
(b)



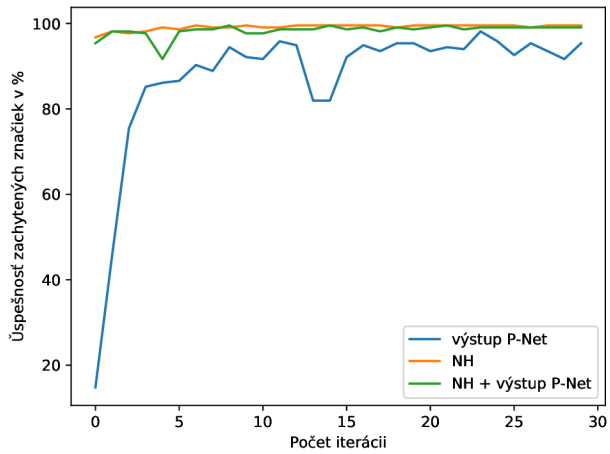
(c)

Obr. 6.2: Grafy z tréningu siete R-Net. Na grafe (a) je zobrazený vývoj chyby, (b) ukazuje vývoj úspešnosti zachytených značiek a (c) znázorňuje pomer medzi zachytením značiek a falošnou detekciou.

Obr. 6.3: Príklady falošne pozitívnych detekcií zo siete R-Net, obsahujúce značky nezahrnuté do klasifikácie.

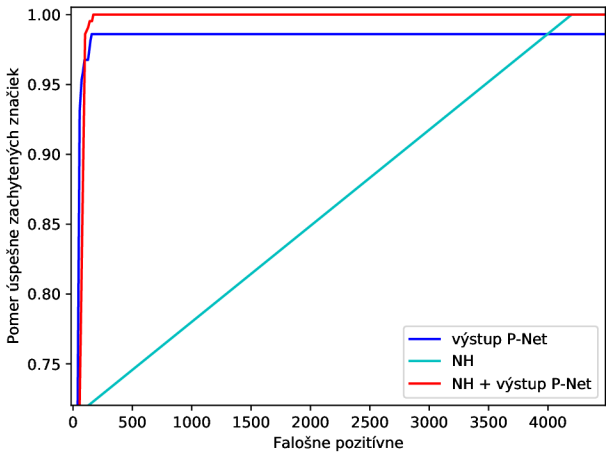


(a)



(b)

Obr. 6.4: V grafoch (a) a (b) sú zobrazené trénovacie behy siete R-Net na 3 rôznych dátových sadách. V legende skratka `NH` predstavuje priebeh trénovania na dátovej sade s náhodne generovanými negatívami, `výstup P-Net` trénovanie na dátovej sade skladajúcej sa z výstupu prvej siete P-Net a `NH + výstup P-Net` je potom kombinácia týchto dvoch.



Obr. 6.5: Graf zobrazuje ako sa vyvíja pomer medzi falošne pozitívnymi a zachytením značiek, na modeloch nátrénovaných na 3 rôznych dátových sadách. Legendy sú vysvetlené v predchádzajúcom obrázku.

Súhrn

Na základe experimentov zobrazených na grafoch 6.4 a 6.5 môžeme usúdiť, že jednoznačne najlepšie výsledky dosiahla R-Net pri použití dátovej sady, vytvorenej kombináciou výstupu siete P-Net a základnou dátovou sadou. V úspešnosti zachytených značiek jasne zaostávala dátová sada vytvorená z výstupu P-Net a v počte chybných detekcii zasa náhodne generovaná dátová sada. Kombinácia týchto dvoch spája ich silné stránky.

6.3 Tréning tretej siete O-Net

Tretia sieť ako posledná v modeli predikuje triedu jednotlivých boxov. Na základe výsledkov predchádzajúceho experimentu nad sieťou R-Net, som sa rozhodol pre použitie dátovej sady vytvorenej kombináciu NH + výstup siete R-Net.

Parametre tréningu

```
{
  "net" : "third",
  "epochs_num" : 35,
  "batch_size" : 128,
  "class-mode" : "multi",
  "dataset": "rnet_merge",
  "base_lr" : 0.005,
  "max_lr" : 0.015,
  "cycle" : 3,
  "classification_weight" : 1,
  "regression_weight" : 1
}
```

Experiment: váhy chýb

Rovnako ako u predchádzajúcich dvoch sietí som O-Net trénoval s rovnakými váhami chýb, teda 1 pre klasifikačnú chybu a 1 pre chybu regresie ohraničujúcich boxov. Z výsledkov však bolo patrné, nedostatočné zarovnanie niektorých boxov na základe čoho som sa rozhodol porovnať výsledky pre štyri rôzne hodnoty váh chyby regresie ohraničujúcich boxov a to `1, 2, 3, 4`.

	Váha 1	Váha 2	Váha 3	Váha 4
Beh 1	90.28	94.91	96.3	97.22
Beh 2	92.59	95.37	94.91	94.91
Beh 3	90.28	92.59	96.3	94.91
Beh 4	91.67	94.44	95.83	96.3
Priemer	91.20	94.90	95.83	95.83

Tabuľka 6.1: Výsledky 4 tréningov pre každú z váh regresie boxov. Uvedené hodnoty predstavujú najvyššiu dosiahnutú úspešnosť v percentách (zachytenie značky) v rámci konkrétneho tréningu. Hranica pre pozitívnu detekciu značky je v tomto prípade zvýšená na 0.75 (IOU).

Súhrn

Pre IOU > 0.65 bol rozdiel medzi jednotlivými behmi zanedbateľný, väčší rozdiel sa ukázal až po zvýšení hranice na 0.75. Pre každú hodnotu som nechal bežať 4 tréningové behy. Výsledky sú zobrazené v tabuľke 6.1. Ako vyrovnané sa ukázali trénovacie behy s hodnotami váhy `3` a `4`, ktoré v priemere dosiahli totožné výsledky, ostatné výrazne zaostávali.

Následne som použil hodnotu `3` i pri bežnom tréningu (IOU > 0.65) a najlepšie natrénovaný model dosiahol úspešnosť 98,8 % (metrika f1).

6.4 Vyhodnotenie modelu

Pre správnu predstavu o fungovaní modelu je potrebné ho otestovať na vyhodnocovacích dátach. Tým je myslená množina dátových vstupov, s ktorými sa model v rámci tréningu nestretol. Z toho dokážeme zistiť jeho správanie na nepozorovaných vstupoch. Práve pre tieto účely som ponechal 300 snímkov z detekčnej dátovej sady. K zjednodušeniu sú použité iba obrázky s minimálne jednou značkou, čo skresalo ich počet na 235. Výsledné hodnoty behu:

- z 361 značiek model úspešne detekoval a zaradil 356, čo predstavuje 98.61 %
- počet falošných detekcií 10

Z týchto údajov je vypočítaná hodnota recall

$$0.986 = \frac{356}{361}. \tag{6.4}$$

a hodnota precision

$$0.972 = \frac{356}{366}. \tag{6.5}$$

Na základe týchto dvoch údajov dokážeme vypočítať metriku f1

$$0.978 = 2 * \frac{0.986 * 0.972}{0.986 + 0.972}. \tag{6.6}$$

Výsledný natrénovaný model MTCNN dosahuje presnosť 97.8 %.

V dvoch prípadoch som postrehol, že detekcie vyhodnotené ako chybné v skutočnosti úspešne zachytili značky, ktoré sú príliš malé a preto neboli uvedené v anotácii detekčnej dátovej sady. Jeden z týchto prípadov je možne postrehnúť na obrázku 6.6.

Obr. 6.6: V pozadí je možné vidieť korektne zachytenú značku triedy 13 (daj prednosť v jazde), ktorá sa ale nenachádza v anotácii a teda jej detekcia bola vyhodnotená ako chybná.

Kapitola 7

Záver

Cieľom tejto práce bola implementácia a experimentovanie nad konkrétnym modelom hlbokých neurónových sietí použiteľným k detekcii. Zvolil som si kaskádový model neurónových sietí MTCNN. Avšak narozdiel od pôvodného zamerania (detekcia tvárí) som ho upravil, implementoval a natrénoval tak aby dokázal detekovať dopravné značky. MTCNN sa skladá z troch jednoduchých konvolučných sietí (P-Net, R-Net, O-Net), ktoré som v rámci tejto práce trénoval. Najväčší problém bol s dátami, žiadna dátová sada totiž nespĺňala všetky moje požiadavky. Nakoniec som použil kombináciu dvoch dátových sád, GTSRB a GTSDB, z nich boli vygenerované trénovacie dáta. Model ako celok v tomto prípade rieši klasifikáciu 43 tried značiek s tým, že prvé dve siete (P-Net, O-Net) riešia iba binárnu klasifikáciu a roztriedenie je ponechané na poslednej sieti O-Net.

Použitie tejto metódy hodnotím za celkom vydarené, obe zo sietí P-Net a R-Net dokázali v rámci tréningu zachytiť všetky dopravné značky, čo sa odzrkadlilo na výsledku celého modelu. Pri vyhodnocovaní na dátovej sade GTSDB dosiahol model úspešnosť 97,8 %.

Cieľ budúceho vývoja vidím hlavne v optimalizácii kódu a s tým spojenú podpore behu na grafických kartách. Ohľadom presnosti modelu by som videl ďalší krok k zlepšeniu výsledkov obstaraním väčšej dátovej sady celých obrázkov z reálnych ciest s vyrovnanou distribúciou tried. Nepochybujem, že týmto by bolo možné dosiahnuť ešte väčšej presnosti.

Literatúra

[1] Alex, K.; Ilya, S.; Hg, E.: *Imagenet classification with deep convolutional neural networks. Proceedings of NIPS, IEEE, Neural Information Processing System Foundation*, Január 2012.

[2] Dertat, A.: *Applied Deep Learning - Part 1: Artificial Neural Networks*. August 2017, [Online; navštívené 15.03.2019].
URL https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

[3] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016, ISBN 0262035618.

[4] Houben, S.; Stallkamp, J.; Salmen, J.; aj.: *Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark*. In *International Joint Conference on Neural Networks*, 1288, 2013.

[5] Jarosz, Q.: *Neuron Hand-tuned.svg*. [Online; navštívené 14.04.2019].
URL https://en.wikipedia.org/wiki/File:Neuron_Hand-tuned.svg

[6] Karpathy, A.: *Convolutional Neural Networks: Architectures*. [Online; navštívené 20.04.2019].
URL http://cs231n.github.io/convolutional-networks/

[7] Karpathy, A.: *Linear Classification*. [Online; navštívené 10.04.2019].
URL http://cs231n.github.io/linear-classify/

[8] Karpathy, A.: *Neural Networks Part 1: Setting up the Architecture*. [Online; navštívené 10.04.2019].
URL http://cs231n.github.io/neural-networks-1/

[9] Lihua, W.; Jo, K.-H.: *Traffic sign recognition and classification with modified residual networks*. December 2017, s. 835–840, doi:10.1109/SII.2017.8279326.

[10] Rosebrock, A.: *Intersection over Union (IoU) for object detection*. November 2016, [Online; navštívené 14.04.2019].
URL https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[11] Smith, L. N.: More Pesky Learning Rate Guessing Games. *CoRR*, ročník abs/1506.01186, 2015.
URL http://arxiv.org/abs/1506.01186

[12] Stallkamp, J.; Schlipsing, M.; Salmen, J.; aj.: *Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. Neural Networks*, , č. 0, 2012: s. –, ISSN 0893-6080, doi:10.1016/j.neunet.2012.02.016.
URL http://www.sciencedirect.com/science/article/pii/S0893608012000457

[13] Xu, B.; Wang, N.; Chen, T.; aj.: *Empirical Evaluation of Rectified Activations in Convolutional Network. CoRR*, Máj 2015.
URL http://arxiv.org/abs/1505.00853

[14] Zhang, K.; Zhang, Z.; Li, Z.; aj.: Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. Október 2016, [Online; navštívené 08.03.2019].
URL
https://kpzhang93.github.io/MTCNN_face_detection_alignment/index.html

[15] Zhang, K.; Zhang, Z.; Li, Z.; aj.: *Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. IEEE Signal Processing Letters*, ročník 23, č. 10, Október 2016: s. 1499–1503, ISSN 1070-9908, doi:10.1109/LSP.2016.2603342.

[16] Šíma, J.; Neruda, R.: *Teoretické otázky neuronových sítí*. Praha: MATFYZPRESS, prvé vydanie, 1996, ISBN 80-85863-18-9.

Príloha A

Obsah priloženého pamäťového média

Readme.txt V tomto súbore je bližšie popísaná štruktúru jednotlivých zložiek.

thease.pdf Text bakalárskej práce vo formáte pdf.

thease Zložka obsahujúca zdrojové súbory textu bakalárskej práce vo formáte LaTeX.

data V tejto zložke sú umiestnené dátové sady použité k učeniu sietí.

code Zložka obsahujúca všetky zdrojové súbory implementované v rámci bakalárskej práce.

plagat.pdf Plagát prezentujúci výsledky práce.