FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Analýza pohybu pro staticky umístěnou kameru
Student:	Matej Matula
Vedoucí:	Ing. Radomír Polách
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Nastudujte knihovny OpenCV [1] a/nebo SimpleCV [2] a patřičné algoritmy a jejich použití v analýze pohybu pro staticky úmístěnou kameru.
1) Navrhněte vhodné použití algoritmů pro detekci a sledování pohybu humanoidů ve video záznamu.
2) Navrhněte vhodné použití algoritmů pro detekci a sledování nehumanoidních objektů.
3) Navrhněte vhodné použití algoritmů pro analýzu míst s největším množstvím pohybu (heat mapy).
Na základně návrhů vytvořte funkční prototypy aplikací pracující pod operačními systémy Linux a Windows. Navržené aplikace otestujte s vhodnými video záznamy.

Seznam odborné literatury

[1] OpenCV. Dostupné z: http://opencv.org/
[2] SimpleCV. Dostupné z: http://simplecv.org/

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 24. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Analýza pohybu pre staticky umiestnenú kameru

Matej Matula

Katedra Softwarového Inžinierstva
Vedúci práce: Radomír Polách

15. mája 2018

Poďakovanie

Rád by som sa poďakoval pánovi Ing. Radomírovi Poláchovy za vedenie práce a čas strávený nad konzultáciami a diskusiami na danú tému.

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 15. mája 2018 ……

Odkaz na túto prácu

Matula, Matej. *Analýza pohybu pre staticky umiestnenú kameru.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Cieľom práce bolo vytvoriť aplikáciu na detekovenie ľudí a nehumanoidných objektov, a sledovanie ich pohybu pomocou staticky umiestenej kamery. Aplikácia využíva konvolučné neurónové siete, knižnicu OpenCV na efektívnu manipuláciu s obrázkami, framework CUDA pre využitie GPU na výpočetné prvky a framework KERAS pre neurónové siete. Vysledná aplikácia by mala byť schopná sledovať ľudí a nehumanoidné objetky, a ďalej spracovať ich pohyb do tepelných máp. Výsledná aplikácia bude môcť byť nasadená v supermarketoch pre marketingové účely.

Klíčová slova Počítačové videnie, Umelá Inteligencia, YOLO, Konvolučna neurónová sieť, OpenCV, CUDA, Tepelná mapa

Abstract

The main target of this Thesis was to create application to detect people and non humanoid objects and track their movement with help of a static positioned camera. The application uses convolutional neural networks, the OpenCV library for effective imagines manipulation, the CUDA framework for GPU usage on computational elements and the KERAS framework for neural networks. The final aplication should be able to track people and non humanoid objects and process their movement into heat maps. This application can be utilized for marketing purposes by supermarkets.

Keywords Computer vision, Artificial intelligence, YOLO, Convolutional neural network, OpenCV, CUDA, Heat Map

Obsah

Úvod **1**

1 Cieľ práce **3**

2 Popis problému **5**
2.1 Súčasný stav riešenia problematiky 5
2.2 Existujúce možnosti riešenia 5
2.3 Vyčlenenie pojmov 6

3 Analýza a návrh **19**
3.1 Analýza požiadaviek 19
3.2 Analýza problému 20
3.3 Návrh tried 21

4 Technológie **25**

5 Realizácia **27**
5.1 Predspracovanie dát a datasety 27
5.2 Doplnenie dát 28
5.3 Vytvorenie dát 28
5.4 Aplikácia na labelovanie dát 29
5.5 USC pedestrain dataset 30
5.6 PennFudanPen dataset 32
5.7 Architektúra siete 32
5.8 Trénovanie 33
5.9 Tepelné mapy 41

Záver **43**

Literatúra **45**

A Zoznam použitých skratiek **49**

B Obsah priloženého CD **51**

Zoznam obrázkov

2.1 Neuron. 7
2.2 Ukážka doprednej siete. 8
2.3 Ukážka rekurentnej siete. 8
2.4 Ukážka hlbokej siete. 9
2.5 Ukážka grafu funkcie Sigmoid. 10
2.6 Ukážka grafu funkcie ReLU. 11
2.7 Ukážka grafu funkcie LeakyReLU. 11
2.8 Ukážka grafu fukncie Tanh. 12
2.9 Ukážka CNN. 13
2.10 Ukážka konvolúcie pre vstup o veľkosti $5 \times 5 \times 1$ a kernel veľkosti $3 \times 3 \times 1$. 14
2.11 Ukážka výsledku pre vstup o veľkosti $5 \times 5 \times 1$ a kernel veľkosti $3 \times 3 \times 1$. 14
2.12 Ukážka paddingu 2 pre vstup o veľkosti $32 \times 32 \times 3$. 15
2.13 Ukážka maxpoolu. 16
2.14 Ukážka averagepoolu. 16

3.1 Class diagram parserov. 22
3.2 Class diagram generatorov. 23

5.1 Ukážka normálneho a otočeného obrázka. 29
5.2 Ukážka diagramu aktivít. 30
5.3 Ukážka grafu straty pre prvotné trénovanie. 33
5.4 Ukážka výsledku prvotného trénovania. 34
5.5 ukážka problémových miest prvotného trénovania 34
5.6 Ukážka grafu straty pre trénovanie na čiernobielých obrázkoch. . . 35
5.7 Ukážka IOU. 36
5.8 Ukážka grafu straty pri použití IOU. 37
5.9 Ukážka grafu straty pri použitia MSE. 37

5.10 Ukážka grafu straty pri trénovaní na obrázkoch, z ktorých bolo odstránené pozadie. 38
5.11 Ukážka výsledku trénovania na odstránenom pozadí. 38
5.12 Ukážka výsledku algoritmu yolo. 39
5.13 Ukážka výsledku trénovania na autách. 39
5.14 Ukážka výsledku trénovania na autách. 40
5.15 ukážka výsledku na obrázku z iného datasetu 40
5.16 Ukážka detekcie na jednom aute. 41
5.17 Ukážka farebnej schémy. 42
5.18 Ukážka tepelnej mapy. 42

Úvod

Umelá inteligencia je v poslednej dobe najviac rastúcim odborom informačných technológií. Počítačové videnie (Computer vision) je podoblasť umelej inteligencie, ktorá sa dá považovať za jej najviac rastúcu časť. S vývojom hardwaru a zvyšovaním výpočetnej sily, dokážeme obrázok klasifikovať, vyťažiť z neho dáta, alebo detekovať niekoľko objektov z obrázka vo veľmi krátkej dobe. Moderný hardware umožnil vytvorenie nových metód a algoritmov, vďaka ktorým vieme spracovať dáta aj v reálnom čase z kamery. Veľkú zásluhu na tom majú hlavne neurónové siete. Práve im sa vďaka spomínaným veciam pripisujú veľké zásluhy v počítačovom videní. Od roku 2012, kedy v každoročnej súťaži v počítačovom videní dominovalo riešenie [1] s chybovosťou 15%, ktoré používalo hlboké učenie, teda niekoľkovrstvovú neurónovú sieť, sa stali neuronové siete, hlavne konvolčuné neurónové siete, ktoré počítajú s tým, že na vstupe je obrázok neoddeliteľnou súčasťou tejto kategórie. Vznikli a stále vznikajú nové frameworky, ktoré ponúkajú istý level abstrakcie nad neurónovými sieťami, čo ponúka možnosť okamžite sa vrhnúť na výskum a prácu a vynechať tak implementačné detaily. Vďaka týmto pokrokom vieme detekovať pohyb osôb, alebo dopravné značky, čo nám umožňuje vytvárať napríklad bezpečné samoriadiace autá [3], perspektívu detekovať podozrivé osoby a predmety, a tým začať novú éru bezpečnosti a komfortu pre ľudstvo.

Kapitola 1

Cieľ práce

Cieľom tejto práce je navrhnúť architektúru konvolučnej neurónovej siete a zhotoviť prototyp aplikácie pre detekciu osôb alebo nehumanoidných objektov zo záznamu staticky umiestnenej kamery, videa, obrázka a ďalej zo získaných dát zkonštruovať tepelnú mapu. Výstupom tejto aplikácie by mal byť upravený vstup aplikácie s obdĺžnikom okolo každej detekovanej osoby alebo nehumanoidného objektu, podľa požiadavky.

Kapitola 2

Popis problému

2.1 Súčasný stav riešenia problematiky

V súčasnosti existuje kamera, no nie je na jej záznam nasadená žiadna aplikácia, ktorá by sledovala ľudí. Tak isto vo vybraných a oslovených supermarketoch existuje kamerový systém, no nevyužíva žiadnu aplikáciu s touto funkcionalitou, čo by umožnilo získavanie marketingových dát.

2.2 Existujúce možnosti riešenia

2.2.1 YOLO - You Only Look Once

Do dnešnej doby je najúčinnejší algoritmus pre detekovanie objektov na videu, obrázku alebo v real time algoritmus YOLO [20] - You Only Look Once. Algoritmus je naimplementovaný vo frameworku Darknet. Od jeho prvého vzniku boli vytvorené ďalšie 2 verzie, každá lepšia, rýchlejšia a presnejšia ako predchádzajúca. YOLO rozdelí vstup, teda obrázok na $13 \times 13 \times 30$ grid. Každé okno je zodpovedné za detekciu 2 boxov, každý box je reprezentovaný súradnicou x, súradnicou y, výškou, šírkou a pravdepodobnosťou, že sa v ňom niečo nachádza. Ďalších 20 prvkov reprezentuje počet klás, teda pravdepodobnosť, že objekt reprezentuje danú triedu.

2.2.2 Floating Window implementation

Jednou z možných implementácii je Floating Window, teda plávajúce okno. Spočíva v natrénovaní neurónovej siete pre klasifikáciu obrázku podľa požadovanej kategórie. Následne sa vytvorí okno o veľkosti $N \times N$, ktoré sa pohybuje po celom obrázku. Začína v ľavom hornom rohu, pri každom pohnutí sa posunie o predom určenú časť do prava.

Ak zaeviduje koniec osi x, vráti sa na jej začiatok, no posunie sa o predom určenú časť na osi y. Toto sa opakuje, až kým okno nepokryje každý pixel

obrázku. Po dokončení sa môže veľkosť okna zmeniť a celý cyklus zopakovať. Každá iterácia pošle do neurónovej siete časť obrázku, ktorú práve pokrýva okno ako vstup. Ak sieť vyhodnotí, že sa na obrázku nachádza požadovaný objekt, rozmery a polohu okna si uloží. Po prejdení všetkých častí, sa uložené časti spracujú pomocou algortimov (napríklad max suppression) a vykreslia sa vypočítané obdĺžniky do obrázku.

2.3 Vyčlenenie pojmov

2.3.1 Umelá inteligencia

Podľa otca umelej inteligence, Johna McCartyho, „Umelá inteligencia je veda a inžinierstvo, ktoré vytvárajú inteligentné stroje, obzvlášť inteligentné počítačové programy.Ïnými slovami, je to snaha naučiť program rozmýšľať a rozhodovať sa ako človek. Hlavnou inšpiráciou pri výskume umelej inteligencie je ľudský mozog, ktorého činnosť sa vedci snažia skúmať a napodobniť. Umelá inteligencia sa rozdeľuje na viacero odvetí, ktoré sa zaoberajú inou činnosťou, napr. plánovanie, klasifikácia, detekcia, rozhodovanie.Avšak, každá umelá inteligencia má jedno zameranie, na ktoré je určená. Práve najhlavnejším cieľom výskumu umelej inteligence je vytvorenie takzvanej Všeobecnej umelej inteligencie, ktorá by dokázala simulovať dokonale správanie človeka, a teda by si dokázala poradiť s každým problémom, s ktorým by sa stretla.

Umelá inteligencia je terčom rôznych debát, ktoré sa zaoberajú jej dopadom na človeka v budúcnosti. Téma debát prechádza od možnosti nahradiť ľudskú silu umelou inteligenciou, až po strach zo sebauvedomenia umelej inteligencie čo podľa mnohých znamená skazu pre ľudstvo, ak sa v budúcnosti nevykonajú patričné opatrenia, ktoré budú tento scenár znemožňovať.

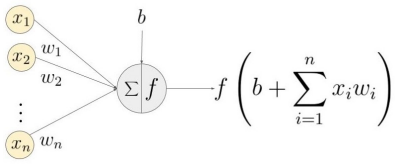
2.3.2 Umelá neurónová sieť

Umelá neurónová sieť je druh umelej inteligencie, ktorý bol namodelovaný na základe štúdia mozgu, presnejšie neurónov. Princíp umelej neurónovej siete je rovnaký ako u tej biologickej. Na základe dát sa učí vykonávať činnosť, na ktorú sa práve trénuje (napríklad detekovať na obrázku psov alebo podľa hlasového záznamu určiť vek volajúceho). Skladá sa zvyčajne z 3 vrstiev: zo vstupnej, skrytej a výstupnej.

Vstupná vrstva predstavuje vstupné dáta. Vstupnými dátami môže byť napríklad obrázok alebo sekvencia dát. Skrytá vrstva sa stará o transformáciu dát, ktoré potom môže výstupná vrstva vyhodnotiť ako výstupné dáta. Každá vrstva sa skladá z neurónov (buniek), vrstvy sú poprepájané váhami, ktoré vedú z každého neurónu jednej vrstvy, do každého neurónu druhej vrstvy. Hodnota neurónov vstupnej vrstvy predstavuje hodnotu vstupných dát. V prípade obrázku, každý neurón drží hodnotu jedného pixelu. Hodnota neurónov v ostatných vrstvách sa vypočíta ako suma hodnôt všetkých neurónov z

predchádzajúcej vrstvy násobené váhou, ktorou sú spojené s neurónom, ktorého hodnotu chceme vypočítať a pripočítaním biasovej hodnoty (pomocou ktorej vieme posúvať hodnotu, a tým ovplivniť aktiváciu) neurónu. Výsledné číslo sa ďalej vloží ako vstup do aktivačnej funkcie a jej výstup predstavuje hodnotu neurónu nazývaný aj aktivácia.

Ukážka neurónu:



Obr. 2.1: Neuron.

Source: www.learnopencv.com

Na obrázku vyššie vidíme:

$x_1 ... x_n$ sú vstupy neurónu, teda neuróny predchádzajúcej vrstvy.

$w_1 .... w_n$ sú váhy. Váha w_i spája neurón x_i s neurónom, ktorého hodnotu počítame.

$f()$ je aktivačná funkcia.

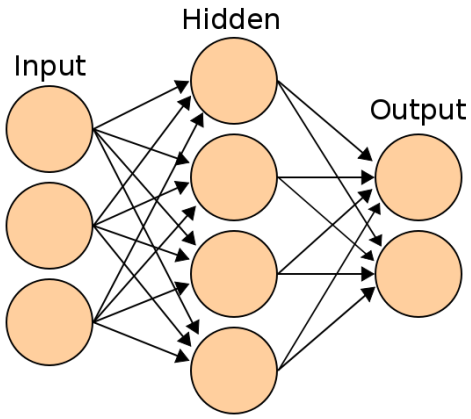
b je bias neurónu.

Šikovným spôsobom, ako efektívne vypočítať hodnoty neurónov ďalšej vrstvy je použiť maticové násobenie a zaobchádzať s váhami a hodnotami neurónov ako s vektormi:

$$\begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & w_{1,2} & \dots & w_{1,n} \\ \dots & \dots & \dots & \dots & \dots \\ w_{k,0} & w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} * \begin{bmatrix} a_0^{(l-1)} \\ a_1^{(l-1)} \\ a_2^{(l-1)} \\ \dots \\ a_n^{(l-1)} \end{bmatrix}.$$

Na obrázku výššie predstavuje a_i^l i-ty neurón l-tej vrstvy, a $w_{k,n}$ je váha spájajúca n-ty neurón vo vrstve l s k-tym neurónom vo vrstve $l+1$. Takáto neurónová sieť, ktorej výstup z jednej vrstvy je vstupom do druhej vrstvy, sa nazýva aj dopredná, teda neobsahuje žiadne sľučky.

Ukážka doprednej siete:

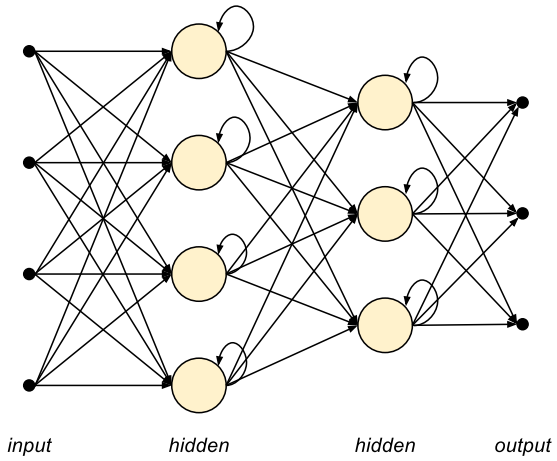


Obr. 2.2: Ukážka doprednej siete.

Source: https://commons.wikimedia.org

Exitujú modely, v ktorých sľučky existujú, tieto siete sa volajú rekurentné neurónové siete.

Ukážka rekurentnej siete:



Obr. 2.3: Ukážka rekurentnej siete.

Source: www.xcelerit.net

Výsledok neurónovej siete sa porovná s požadovaným výsledkom, a na základe určenej stratovej funkcie sa vypočíta strata. Na základe tejto straty sa pomocou backpropagácie upravia váhy a biasi neurónov. Práve nájdenie adekvátnych váh a biasov je cieľom neurónovej siete.

2.3.3 Trénovanie a učenie siete

Naučenie alebo trénovanie neurónovej siete spočíva v úprave váh a biasov na také hodnoty, ktoré po prehnaní vstupu cez neurónovú sieť zabezpečia, že výsledok, ktorý sieť vytvorí, bude odpovedať skutočnosti. Inými slovami je to hľadanie takých váh a biasov, pri ktorých je strata vyhodnotená stratovou funkciou minimálna.

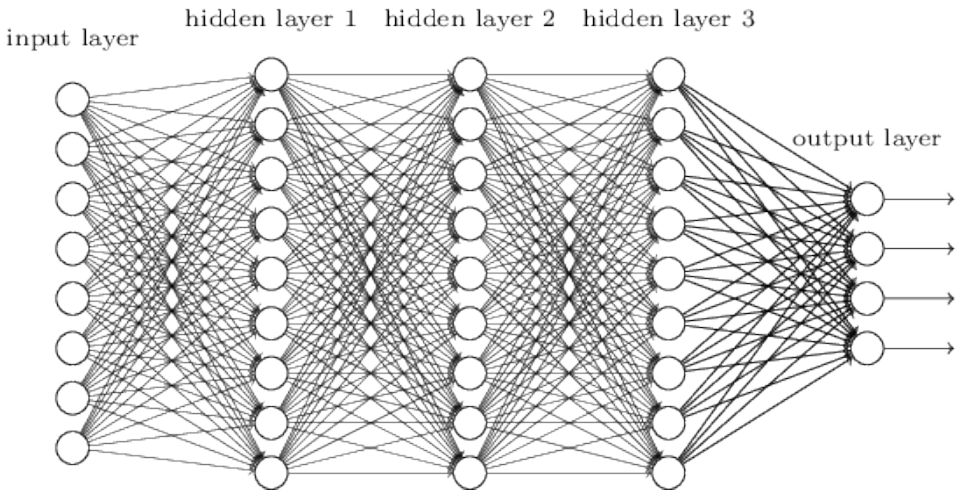
Úprava váh a biasov sa vykonáva pomocou algoritmu Backpropagation. Na začiatku sa všetky váhy a biasy nadstavia na náhodné hodnoty. Po získaní výsledku siete sa vypočíta strata pomocou stratovej funkcie a na záver sa určí, ktoré váhy sa majú zmeniť a o koľko.

Celý proces funguje na základe Gradient Descent, a hľadania záporného gradientu, teda hľadanie minima vo funkcií a výpočet parciálnych derivácii. Gradient Descent a Backpropagation sú rozsiahle pojmy, pre ktoré popisy by bolo možné zhotoviť ďalšiu bakalársku prácu, preto ich nebudeme podrobne popisovať. Informácie o nich sa nachádzajú v referencií [21].

2.3.4 Hlboká umelá neurónová sieť

Hlboká umelá neurónová sieť je viac vrstvová klasická umelá neurónová sieť. Obsahuje viac skrytých vrstiev, čo umožňuje naučenie sa viac špecifických prvkov z vložených dát. Neplatí však priama úmernosť. Väčší počet vrstiev nezaručuje väčšiu presnosť siete. Z teoretického hľadiska, nie je dokázané, že hlboká sieť je presnejšia ako obyčajná (alebo plytká) sieť, aj keď z praktického hľadiska, hlboké siete vykazuju lepšie a presnejšie výsledky.

Ukážka hlbokej sieti:



Obr. 2.4: Ukážka hlbokej siete.

Source: http://neuralnetworksanddeeplearning.com

2.3.5 Aktivačná funkcia

Aktivačná funkcia je spôsob, ktorým môžeme normalizovať vstup na menšie prehľadnejšie hodnoty, kedže neurón samotný nevie určiť hranice hodnôt. V umelých neurónových sieťach sa používa na výpočet hodnoty neurónu. Vďaka tejto hodnote vieme určiť, či bude neurón aktívny alebo nie.

Rozlišujeme 2 základné druhy aktivačných funkcí:

1. **Lineárna aktivačná funkcia.**
2. **Nelineárna aktivačná funkcia.**

2.3.6 Lineárna aktivačná funkcia

Jednoduchá funkcia, ktorá transformuje vstup pomocou vzorca:

$$f(x) = x.$$

2.3.7 Nelineárna aktivačná funkcia

Komplexnejšia, častejšie používaná funkcia, ktorá sa používa na dáta, kde nie je možné použiť lineárnu funkciu, alebo je to neefektívne. Rozlišujeme niekoľko druhov.

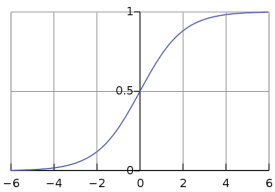
2.3.7.1 Sigmoid

Sigmoid funkcia je ohraničená, diferencovateľná, reálna funkcia, ktorá je definovaná pre všetky reálne vstupy a v každom bode má nezápornú deriváciu. [2] Je definovaná predpisom:

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}.$$

Tvar Sigmoid funkcie je krivka v tvare S. Definičný obor funkcie je celá množina reálnych čísiel. Obor hodnôt funkcie je $(0, 1)$.

Ukážka grafu funkcie:



Obr. 2.5: Ukážka grafu funkcie Sigmoid.

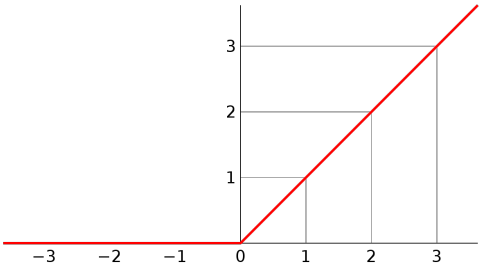
Source: wikipedia.org

2.3.7.2 Rectified Linear Unit (ReLU)

ReLU je nelineárna funkcia, ktorá vracia pozitívnu časť argumentu, teda vráti 0 ak je argument záporný, inak vráti jeho hodnotu. Je definovaná predpisom:

$$f(x) = x^+ = max(0, x).$$

Ukážka grafu funkcie:



Obr. 2.6: Ukážka grafu funkcie ReLU.

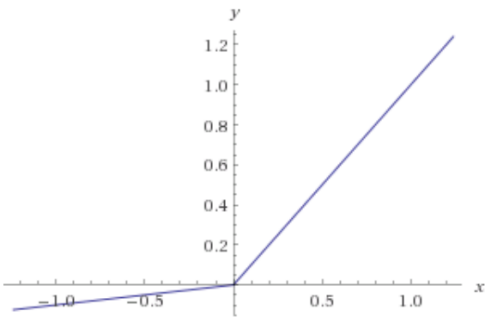
Source: wikipedia.org

Problém s ReLU je stav, ktorý sa volá mŕtve ReLU. Vtedy je výsledok funkcie vždy rovnaký, 0. Tento stav zabraňuje neurónovej sieti sa učiť.

2.3.7.3 Leaky ReLU

Problém mŕtveho ReLU sa snaží vyriešiť Leaky ReLU, ktorý má všetky vlastnosti ReLU, no je definovaný predpisom:

$$f(x) = max(0.1x, x).$$



Obr. 2.7: Ukážka grafu funkcie LeakyReLU.

Source: datascience.stackexchange.com

2.3.7.4 Tanh

Tanh je hyperbolická tangens funkcia [4], ktorá sa vypočíta ako podiel hyperbolického sínusu(sinh) [5] a hyperbolického kosínusu(cosh) [6].

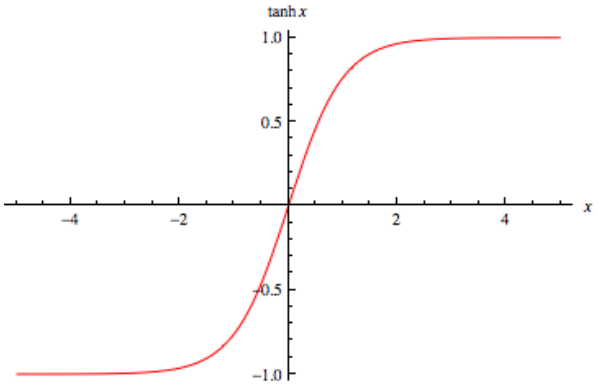
Ukážky vzorcov pre výpočet:

$$sinhx = \frac{e^x - e^{-x}}{2} = \frac{e^{2x} - 1}{2e^x} = \frac{1 - e^{-2x}}{2e^{-x}},$$

$$coshx = \frac{e^x + e^{-x}}{2} = \frac{e^{2x} + 1}{2e^x} = \frac{1 + e^{-2x}}{2e^{-x}},$$

$$tanhx = \frac{sinhx}{coshx} = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Ukážka grafu funkcie:



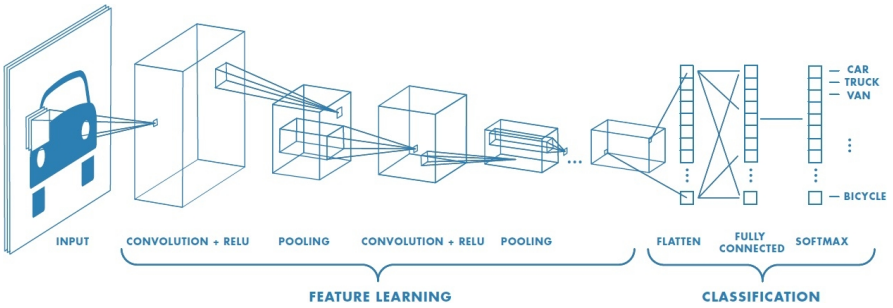
Obr. 2.8: Ukážka grafu fukncie Tanh.

Source: http://mathworld.wolfram.com

2.3.8 Konvolučná neurónová sieť

Konvolučná neurónová sieť je druh hlbokej umelej neurónovej siete, ktorá predpokladá, že na vstupe siete bude obrázok, čo umožňuje postaviť štruktúru siete tak, aby sa mohla naučiť detaily a vzory obsahu obrázkov, využiť ich vlastnosti a tým pádom zmenšiť počet parametrov na učenie.

Na rozdiel od klasickej umelej neurónovej siete, skladá sa z viacerých vrstiev ktoré manipulujú obsahom vstupu. Najčastejšie je to vrstva konvolučná, poolingová a úplná. Od ich predstavenia LeCun et, al.,1989 [7] v 90tich rokov 20 storočia, konvolučné neurónové siete demonštrovali excelentný výkon pri úlohách, ako napríklad klasifikovanie písaných čísel a klasifikácie tváre. [8] Za ich úspechom a progresom stojí hlavne: (i) dostupnosť väčších a väčších



Obr. 2.9: Ukážka CNN.

Source: www.mathworks.com

trénovacích setov s miliónmi označených príkladov; (ii) lepšie a lepšie GPU implementácie; (iii) lepšie stratégie regulovania modelov, ako napríklad Dropout. [9]

Vstup, teda obrázok vidí sieť ako $N \times M \times K$ pole čísel, kde N značí šírku, M značí výšku a k značí počet kanálov obrázku teda RGB. Všetky čísla sú v rozmedzí 0 až 255.

2.3.8.1 Konvolučná vrstva

Táto vrstva je základ každej konvolučnej neurónovej siete. Jej účelom je nájsť (naučiť sa nájsť) špecifické črty vo vstupe. Parametre tejto vrstvy predstavujú filtre.

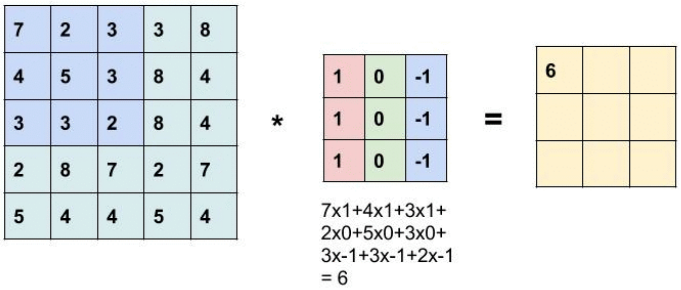
Typicky má filter veľkosť $f \times f \times k$ kde f je typicky celé nepárne číslo a k je hĺbka vstupu. Veľkosť všetkých filtrov v jednej konvolučnej vrstve je rovnaká, čo však neplatí o ich váhach (alebo lepšie povedané, hodnoty, z ktorých sa jednotlivé filtre skladajú). Filtre sú posúvané pozdĺž celého obrázka. Pomocou filtrov sa vykonáva konvolúcia, ktorú môžeme chápať ako váženú sumu medzi regiónom obrázku a filtrom. Z obrázka sa vyberie región o veľkosti filtra a vypočíta sa jeho konvolúcia. Výsledok sa uloží do aktivačnej mapy, ktorá potom slúži ako vstup do ďalšej konvolučnej vrstvy.

Región sa ďalej posunie o parameter Stride na osi x a znova sa vypočíta konvolúcia. Keď sa narazí na koniec osi x, vráti sa na jej začiatok a posunie sa o parameter Stride na osi y, až kým sa nepokryje celý obrázok. Činnosť sa opakuje pre všetky filtre.

Tvar výslednej aktivačnej mapy pre vstup o veľkosti $N \times N \times k$ a kernelu o veľkosti $f \times f \times k$ a Stride S je

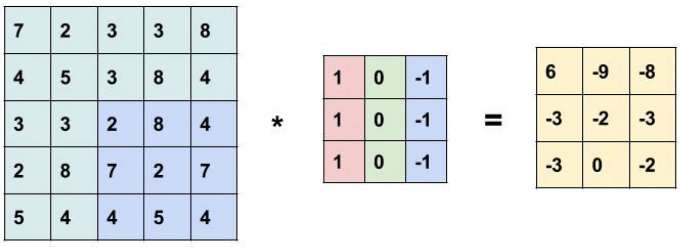
$$(N \times N \times k) * (f \times f \times k) = (\frac{N-f}{S} + 1 \times \frac{N-f}{S} + 1 \times k').$$

kde k' predstavuje počet filtrov.



Obr. 2.10: Ukážka konvolúcie pre vstup o veľkosti $5 \times 5 \times 1$ a kernel veľkosti $3 \times 3 \times 1$.

Source: learnopencv.com



Obr. 2.11: Ukážka výsledku pre vstup o veľkosti $5 \times 5 \times 1$ a kernel veľkosti $3 \times 3 \times 1$.

Source: learnopencv.com

2.3.8.2 Padding

Nežiadúce vlastnosti kovnolúcie sú:

1. **Zmenšovanie vstupu.**

2. **Fakt, že sa krajné pixely, na rozdiel od ostatných, vyskytujú iba v 1 konvolúcii.**

Obom sa dá zabrániť Paddingom. Padding je zmena rozmerov vstupu prídaním ďalších vrstiev do vstupu. Pridané hodnoty vo vstupe sú nadstavené na 0.

Po pridaní paddingu p má výsledná aktivačná mapa rozmery:

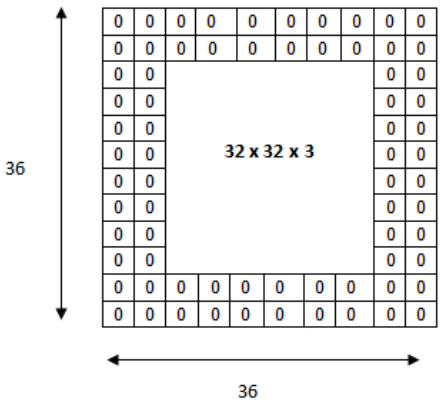
$$\frac{(n+2p-f)}{S}+1 \times \frac{(n+2p-f)}{S}+1 \times k.$$

Najpoužívanejšími typmi paddingu sú:

1. **Valid** - Teda žiadny padding sa nepridáva

2. **Same** - Padding, vďaka ktorému má výsledok konvolúcie rovnaký tvar ako vstup pred pridaním paddingu. Padding P, ktorý je potrebný v tomto prípade pridať do vstupu sa vypočíta pomocou:

$$p = \frac{f-1}{2}.$$



Obr. 2.12: Ukážka paddingu 2 pre vstup o veľkosti $32 \times 32 \times 3$.

Source: adeshpande3.github.io

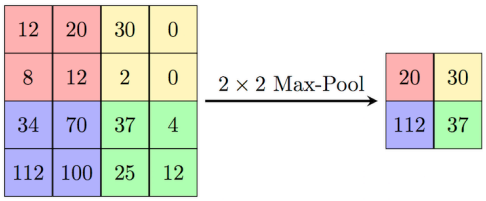
2.3.8.3 Poolingová vrstva

Účelom Poolingových vrstiev je dosiahnuť priestorovú invarianciu znížením priestorového rozlíšenia aktivačných máp. Každá poolovaná aktivačná mapa zodpovedá jednej aktivačne mape predchádzajúcej vrstvy. Ich jednotky kombinujú vstup z malého $n \times n$ regiónu jednotiek. Toto poolingové okno môže byt svojvoľnej veľkosti a okná s môžu prekrývať [10].

2 najpoužívanejšie typy Poolignových vrstiev sú:

1. **Max pooling** - ktorá z regiónu vyberie iba najväčšiu hodnotu. Je definovaná predpisom:

$$h_j^n(x,y) = max_{\overline{x} \in N(x), \overline{y} \in N(y)} h_j^{n-1}(\overline{x}, \overline{y}).$$

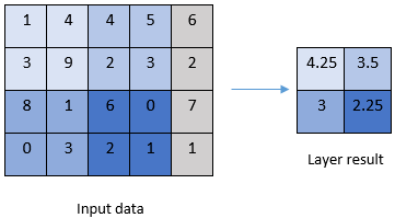


Obr. 2.13: Ukážka maxpoolu.

Source: *https* : *//blog.csdn.net/qq*$_2$6898461/*article/details/*51207376

2. **Average pool** - ktorá z regiónu vypočíta priemernú hodnotu. Je definovaná predpisom:

$$h_j^n(x,y) = 1/K \sum_{\overline{x} \in N(x), \overline{y} \in N(y)} h_j^{n-1}(\overline{x}, \overline{y}).$$



Obr. 2.14: Ukážka averagepoolu.

Source: software.intel.com

2.3.8.4 Výpadková vrstva

Preučenie siete je stav, kedy sieť vykazuje dobré výsledky na trénovacích dátach, no zlé na testovacích. Inak povedané, sieť sa naučí vzory na trénovacích dátach, ale nie generálne vzory. Spôsobov, akým sa táto situácia dá vyriešiť je niekoľko. Jedným z nich je výpadková vrstva. Táto vrstva pri trénovaní náhodne vypne niekoľko neurónov, čo prinúti naučiť sa sieť vyhodnotiť rovnaký výsledok, ale s inými neurónmi. Výpadková sieť je deaktivovaná pri testovaní.

2.3.8.5 Aktivačná vrstva

Aktivačná vrstva slúži na znormalizovanie hodnôt predchádzajúcej vrstvy. Využíva jednu z aktivačných funkcií, ktoré sú definované v sekcii 2.3.5.

2.3.8.6 Úplná vrstva

Všetky neuróny z úplnej vrstvy sú pospájané s neurónmi z predchádzajúcej vrstvy. Je to totožná vrstva pri klasickej umelej neurónovej sieti.

2.3.9 Strátová funkcia

Overenie alebo vyhodnotenie modelu, teda ako ďaleko od skutočného výsledku(y) sa nachádza výsledok, ktorý vyhodnotila neurónová sieť($\hat{y}$) je dôležitou súčasťou učenia, ktoré slúži spolu s backpropagáciou na poupravenie váh siete, teda jej učenie. Táto informácia sa počíta práve pomocou Strátovej funkcie. V súčasnosti je niekoľko populárnych strátových funkcií.

2.3.9.1 Mean squared error(MSE)

Jedna z najpoužívanejších strátových funkcí definovaná predpisom:

$$\mathcal{MSE}(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2,$$

kde $x = \{x_i | i = 1, 2, , N\}$ a $y = \{y_i | i = 1, 2, , N\}$ sú konečné dlhé množiny hodnôt.

Alebo jej generálnou l_p formou:

$$d_p(x, y) = \left(\sum_{i=1}^{N} |e_i|^p\right)^{1/p}.$$

Medzi jej výhody patrí napríklad:

1. Je jednoduchá. Je bez parametrov a lacná na výpočet, so zložitosťou len jedného násobku a dvoch prídavkov na vzorku. Je tiež bez pamäte, chyba štvorca môže byť vyhodnotená na každej vzorke nezávisle od ostatných vzoriek.

2. Všetky l_p normy sú validné vzdialenosti v metrike R^n, ktoré spĺňajú nasledujúce podmienky:

   a) nezápornosť $d_p(x, y) \geq 0$,

   b) identita $d_p(x, y) = 0$ práve vtedy ako $x = y$,

   c) symetria $d_p(x, y) = d_p(y, x)$,

   d) trojuholníková nerovnosť $d_p(x, z) \leq d_p(x, y) + d_p(y, z)$.

Nevýhodou funkcie je napríklad prípad, ak ako aktivačná funkcia bol zvolený Sigmoid. V takom prípade hodnota pomaly konverguje. [11]

2.3.9.2 Mean Squared Logarithmic Error(MSLE)

Varianta klasickej MSE definovaná predpisom:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(log(y^i + 1) - log(\hat{y}^{(i)} + 1)^2.$$

2.3.9.3 L2

Matematicky podobná funkcia MSE definovaná predpisom:

$$\boldsymbol{\mathcal{L}} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2.$$

2.3.9.4 Mean Absolute Error

Aboslútna hodnota MSE definovaná predpisom:

$$\boldsymbol{\mathcal{L}} = \frac{1}{n}\sum_{i=1}^{n}|y^{(i)} - \hat{y}^{(i)}|.$$

Kapitola 3

Analýza a návrh

3.1 Analýza požiadaviek

Táto sekcia definuje požiadavky na novo vytvorenú aplikáciu. Rozlišujeme niekoľko druhov požiadaviek:

1. **Funkčné** - zaoberajú sa požadovanou funckionalitou zhotovovaného programu.
2. **Nefunkčné** - zaoberajú sa požiadavkami na design, prístupnosť a kladú obmedzenia na prostredie.

3.1.1 Funkčné požiadavky

1. Detekcia humanoidných objektov na obrázku, videu alebo real time zázname a ukladanie miest pohybu.

   Scenár úspešného plnenia:

   i) Užívateľ spustí aplikáciu a určí zdroj, na ktorom sa maju detekovať osoby.

   ii) Systém skontroluje, či daný zdroj existuje.

   iii) Systém spracuje zdroj.

   iiii) Systém detekuje na zdroji humanoidné alebo nehumanoidné objekty.

   iiiii) Systém zobrazí upravený zdroj, na ktorom bude každý deketovaný objekt ohraničený obdĺžnikom.

Výnimky:

i **Zdroj neexistuje:** Aplikácia informuje užívateľa o tom, že zadaný zdroj sa nepodarilo nájsť.

ii **Zdroj sa nedá otvoriť:** Aplikácia informuje užívateľa o tom, že zadaný zdroj je v nepodporovanom formáte.

2. Vykreslenie tepelných máp z uložených miest pohybu.

Scenár úspešného plnenia:

i Užívateľ podá požiadavku na vytvorenie tepelnej mapy.

ii Systém detekuje objekty na pridanom zdroji a uloží si ich polohu.

iii Systém vytvorí tepelnú mapu.

iiii Systém zobrazí tepelnú mapu.

Výnimky:

i **Zdroj neexisuje:**Aplikácia informuje užívateľa o tom, že zadaný zdroj sa nepodarilo nájsť.

3.1.2 Nefunkčné požiadavky

1. Aplikácia musí byť rýchla, aby dokázala fungovať na real time videu z kamery.

3.2 Analýza problému

Cieľom tejto práce je detekovať pohyb humanoidných aj nehumanoidných objektov. Na vytvorenie tejto práce budú použité konvolučné neurónové siete, to znamená že implementácia detekcie jednotlivých objektov bude totožná, jediné čo sa bude líšiť je dátový set, na ktorom sa budú detekcie trénovať.

Keďže konvolučné neurónové siete spadajú pod hlboké učenie, je potrebné veľké množstvo dát, teda obrázkov, na ktorých sa môže sieť natrénovať. Dáta sa však nesmú opakovať, to by mohlo viesť k overfittingu. Dáta je však možné poupravovať (otočiť, zrezať), čím vieme vygenerovať viac dát. Taktiež, bude treba získať dáta vhodné pre požadovaný účel. Primárnym zdrojom dát budú verejné, voľne stiahnuteľne a použiteľne datasety. Nie všetky datasety majú rovnaký formát dát, teda bude potrebné pre každý nový druh formátu dát vytvoriť parser, ktorý požadované dáta vytiahne.

S verejnými datasetmy prichádza problémová situácia, kedy obsahujú chybný obrázok, alebo informácie o obrázku nezahrňujú všetky objekty na obrázku. Tomuto problému sa dá predísť buď pozorným prezretím všetkých obrázkov, čo začína byť problematické pri väčšom počte obrázkov, alebo vytvorením vlastného datasetu, čo je časovo namáhavé. Výsledok, a teda aj presnosť

neurónovej siete bude závisieť aj na kvalite a druhu obrázkov, ktoré sa použijú na jej trénovanie, v ideálnom prípade by sa malo použiť niekoľko stoviek tisíc dát rovnakého alebo podobného typu.

Jedným s problémov bude aj výpočetná sila. Pri väčšom datasete bude za potreby dostatok pamäte. Jednou z možností ako tento problém, v prípade nedostatku pamäti vyriešiť, sú cloudové služby.

3.3 Návrh tried

Prototyp aplikácie nebude náročný na návrh. Bude sa skladať z modulu na trénovanie a modulu prototypu aplikácie, ktorý sa bude spúšťať v aplikácii.

Triedy parserov budú dediť z abstraktnej triedy `abstractParser`, ktorá bude mať abstraktnú metódu predstavujúcu získanie načítaných dát z datasetu, `getTrainingData(data,greyScale,addProbability,normalize)` a metódu `normalizeBox(boxes,x_,y_,addProbability)`, ktorá bude predstavovať znormalizovanie štítkov, v prípade že sa rozlíšenie zmenilo

Ďalej bude trieda obsahovať metódu `normalize(img)`, ktorej parameter bude obrázok, ktorý chceme znormalizovať. Jej návratovou hodnotou bude znormalizovaný obrázok z parametru, metódu `greyScale(img)`, ktorej parameter bude obrázok, jej návratovou hodnotou bude čiernobiely obrázok z parametru, a metódu `fillData(data,addProbability)`, ktorá doplní dáta podľa sekcie 5.2.

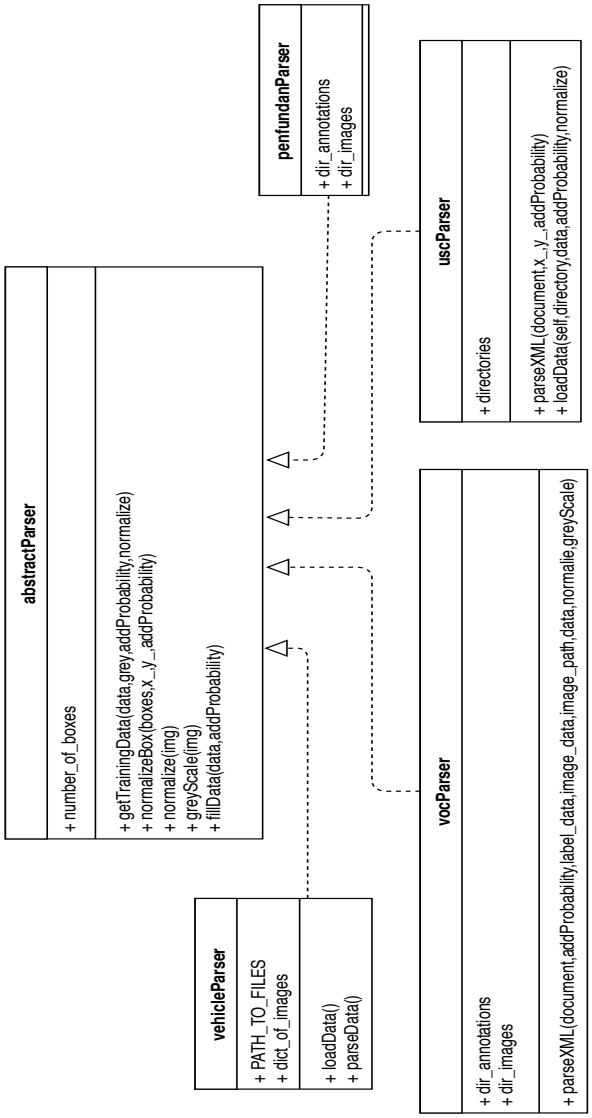
Triedy generátorov budú dediť z abstraktnej triedy `abstractGenerator`, ktorý bude obsahovať abstraktnú metódu predstavujúci získanie načítaných dát `getTrainingData(greyScale,addProbability,normalize,flip)` a metódu `flipData(image_data,label_data)` ktorá otočí obrázky horizontálne a vypočíta novú polohu objektov. Jej návratovú hodnotu predstavujú pole otočených obrázkov a pole nových polôh objektov.

Každý parser bude musieť implementovať metódu svojej abstraktnej triedy `getTrainingData(data,greyScale,addProbability,normalize)`, ktorej parameter `greyScale` bude určovať, či má všetky dáta previesť na čiernobiele alebo nie. Paremter `addProbablity` ktorý bude určovať, či na začiatok každého labelu pridať pravdepodobnosť, či sa v ňom niečo nachádza (0 ak nie, 1 ak áno) a parameter `normalize`, ktorý bude určovať, či sa obrázky majú normalizovať. Dalej budú musieť triedy parserov naimplementovať daľšiu metódu `normalizeBox(boxes,x_,y_,addProbability)`, ktorej parameter `x_` a `y_` bude určovať faktor, o ktorý sa zmenilo rozlíšenie obrázku `addProbability` bude určovať, či na začiatok pridať pravdepodobnosť, či sa v labele niečo nachádza.

Každý generátor dát bude musieť implementovať metódu svojej abstraktnej triedy `getTrainingData(greyScale,addProbability,normalize,flip)`. Kde parameter `greyScale` bude určovať, či jednotlivé parsery, ktoré generátor obsahuje, prevedú dáta na čiernobiele. Parameter `addProbability` bude

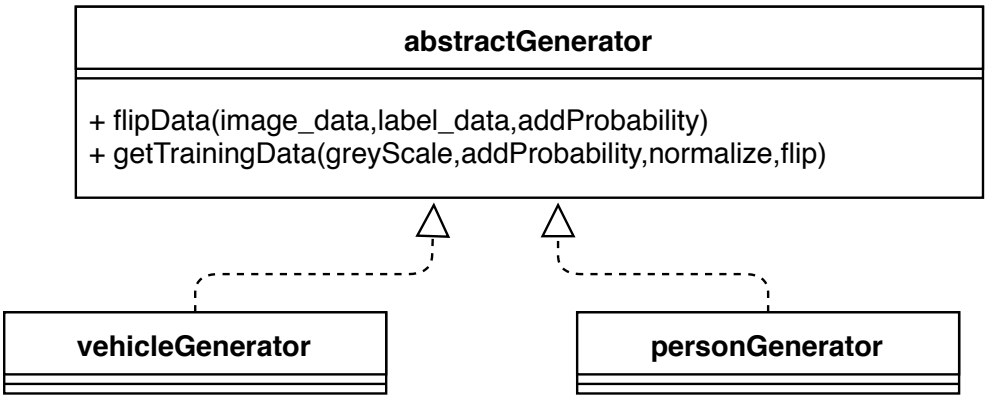
určovať, či parsere pridajú na začiatok štítku pravdepodobnosť, či sa v ňom niečo nachádza, parameter `normalize` bude určovať, či parsere znormalizujú dáta a parameter `flip` bude určovať, či generátor vytvorí dalšie dáta tým, že ich horizontálne otočí.

Ukážka diagramu tried perserov, kde prerušovaná čiara predstavuje dedenie:



Obr. 3.1: Class diagram parserov.

Ukážka diagramu tried generátorov:



Obr. 3.2: Class diagram generatorov.

Kapitola 4

Technológie

Táto kapitola popisuje technológie, ktoré boli využité pri zhotovovaní práce.

4.0.1 Python

Python je dynamicky interpretovaný open-source jazyk. Podporuje 3 základné programovacie paradigmy:

1. procedurálny,
2. funckionálny,
3. objektový.

V dnešnej dobe je python používaný skoro vo všetkých oblastiach computer science.

Python bol zvolený ako hlavný programovací jazyk, kvôli jeho jednoduchosti a silnej podpore v oblasti umelej inteligencie za podoby modulov a frameworkov, ktoré uľahčujú vývoj.

Jednou z variánt výberu jazyka bolo C++, ktoré je taktiež populárnou voľbou pri programovaní umelej inteligencie, no jeho podpora v podobe frameworkov, modulov či knižníc je výrazne horšia. Inštalácia frameworkov potrebných modulov pri tvorbe tejto práce sprevádzala kopa errorov, nekompatibilít a bugov. Z toho dôvodu sa od tejto varianty odstúpilo.

4.0.2 Keras

Keras je high-level API napísané v pythone schopné pracovať nad Tensorflow, CNTK alebo Theano. Bol vyvinutý so zameraním na umožnenie rýchleho experimentovania.

Výhody kerasu:

1. umožňuje jednoduché a rýchle vytváranie prototypov (prostredníctvom užívateľskej prívetivosti, modularity a rozšíriteľností),
2. podporuje konvolučné siete a opakujúce sa siete, ako aj kombinácie týchto dvoch,
3. beží bez problémov na CPU a GPU.

[12]

4.0.3 CUDA

CUDA® je paralelná počítačová platforma a programovací model vyvinutý spoločnosťou NVIDIA pre všeobecné výpočty na grafických procesorových jednotkách (GPU). [13]

Vo výskume umelých neurónových sietí je populárnou voľbou vďaka zrýchleniu výpočtu a tým pádom rýchlejšiemu trénovaniu.

4.0.4 CUDnn

Knižnica NVIDIA CUDA® Deep Neural Network (cuDNN) je zrýchlená knižnica primitívov pre hlboké neurónové siete. CuDNN poskytuje vysoko ladené implementácie pre štandardné rutiny, ako sú dopredné a spätné konvolúcie, združovanie, normalizácia a aktivačné vrstvy. [14]

4.0.5 OpenCV

OpenCV (Open Source Computer Vision) je knižnica programovacích funkcií zameraná hlavne na počítačové videnie v reálnom čase. [15]

Jej výhodou je v efektívnej implementací algoritmov a šikovnej správe pamäti.

4.0.6 Numpy

Numpy je balíček pre python zameraný na vedecké výpočty.

Kapitola 5

Realizácia

V tejto kapitole je popísaný proces tvorby aplikácie.

5.1 Predspracovanie dát a datasety

Dušou každého projektu využívajúci umelé neurónové siete sú dáta. V rámci projektu bolo za potreby získať adekvátny dataset, ktorý odpovedá účelu trénovania. Jednotlivé dáta (obrázky) musia obsahovať rozoznateľné objekty, ktoré sa chcú detekovať a mať ku každému takému objektu, ktorý sa na obrázku vyskytuje informácie o jeho polohe prostredníctvom obdĺžnika reprezentovaný 4 číslami:

1. xmin - poloha horného ľavého rohu na x-ovej osi,
2. ymin - poloha horného ľavého rohu na y-ovej osi,
3. xmax - poloha dolného pravého rohu na x-ovej osi,
4. ymax - poloha dolného pravého rohu na y-ovej osi.

Dáta, ktoré boli použité pri trénovaní vznikli z rôznych zdrojov. Každý zdroj využíval iný formát ukladania oštítkovaných dát, teda bolo za potreby pre každý dataset, ktorý používa iné formátovanie, vytvoriť parser, ktorý požadované dáta vytiahne, poprípade zahodí nepotrebné dáta.

Pôvodné dáta boli v rôznych rozlíšeniach, preto bolo za potreby dáta patríčne pretransformovať. Aplikácia príma na vstupe obrázky o veľkosti:

$$416 \times 416.$$

Pri obrázku odlišnej veľkosti sa najprv veľkosť obrázka upraví a vypočíta x a y factor, teda koeficient o ktorý sa jednotlivá dimenzia zmenila.

Koeficient sa vypočíta ako:

$$x\_factor =< image\_width > /416,$$
$$y\_factor =< image\_height > /416,$$

a vypočíta sa nová poloha obdĺžniku ako:

$$< new\_xmin >=< old\_xmin > *x\_factor,$$
$$< new\_ymin >=< old\_ymin > *y\_factor,$$
$$< new\_xmax >=< old\_xmax > *x\_factor,$$
$$< new\_ymax >=< old\_ymax > *y\_factor.$$

5.2 Doplnenie dát

Rôzne obrázky môžu obsahovať rôzny počet osôb, z čoho vyplíva, že veľkosť štítkového vstupu môže byt rôzna. Aby sa predišlo rôznym veľkostiam, určil sa maximálny počet osôb, ktorý bude sieť detekovať. Jednotlivé inputy sa doplnili východzými hodnotami, aby reprezentovali rovnaký počet osôb. Východzou hodnotou je $[0, 0, 0, 0]$

Ak obrázok obsahoval 3 osoby, a jeho vstupný štítok obsahoval 3 elementy, teda vyzeral ako:

$$[[osoba1], [osoba2], [osoba3]],$$

doplní sa input na:

$$[[osoba1]_1, [osoba2]_2, [osoba3]_3, [0, 0, 0, 0]_4, [0, 0, 0, 0]_5 ... [0, 0, 0, 0]_N].$$

teda aby jeho veľkosť odpovedala počtu maximálne detekujúcich osôb

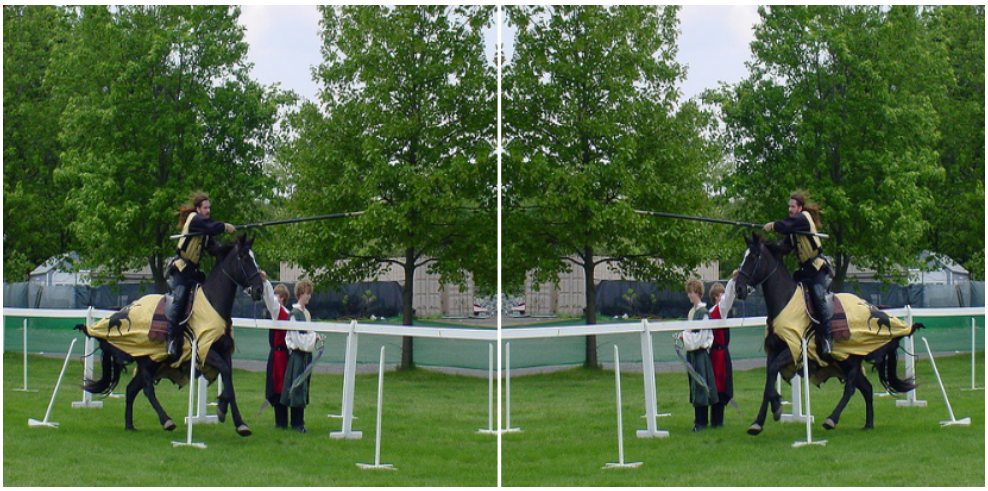
V prípade, že chceme v štítku indikovať pravdepodobnosť, že sa v ňom niečo nachádza, do každého elementu reprezentujúci polohu osoby prídáme na začiatok 1 a východzou hodnotou sa stane $[0, 0, 0, 0, 0]$.

5.3 Vytvorenie dát

Existuje veľa techník, ktorými môžeme z dát, ktoré máme k dispozícii, vytvoriť ďalšie dáta. V tejto aplikácii budeme pre vytvorenie dát používať horizontálne otočenie obrázku.

Ďalšou možnosťou je rotácia obrázku o 90, 180, 270 stupňov. Toto otočenie však indikuje, že objekt môže byť otočený hore nohami alebo bokom. V prípade detekcií objektov by to znamenalo, že sieť sa môže naučiť rozoznávať zle objekt, alebo identifikovať chybný objekt, ako objekt ktorý chceme detekovať. Z toho dôvodu sa v práci ako jediná technika na vytvorenie dát použije horizontálne otočenie obrázku.

Ukážka otočenia obrázku:



Obr. 5.1: Ukážka normálneho a otočeného obrázka.

5.4 Aplikácia na labelovanie dát

Za účelom oštítkovať vlastné dáta, teda dáta ktoré nie sú súčasťou verejného datasetu bola zhotovená jednoduchá Javascriptovská aplikácia na štítkovanie.

Užívateľ klikne na tlačidlo Vybrať súbor, a vyberie obrázok ktorý chce oštítkovať. Následne kliknutím ľavého tlačidla, držaním tlačidla, potiahnutím myšky a uvoľnenia tlačidla vytvorí obdĺžnik/štvorec, okolo objektu ktorého súradnice chce uložiť. Súradnice vytvoreného obdĺžnika sa uložia do pola súradníc.

Pri stlačení ľavého tlačidla myšky si aplikácia uloží x a y súradnice kliknutia, a pri potiahnutí myšky vyčistí plátno na ktoré sa obrázok a obdĺžniky vykresľujú. Vykreslí znova všetky uložené súradnice obdĺžnikov, a vykreslí ďalší obdĺžnik, ktorého ľavý horný roh predstavujú uložené súradnice a jeho šírka a výška sa vypočíta zo súčasnej polohy myšky.

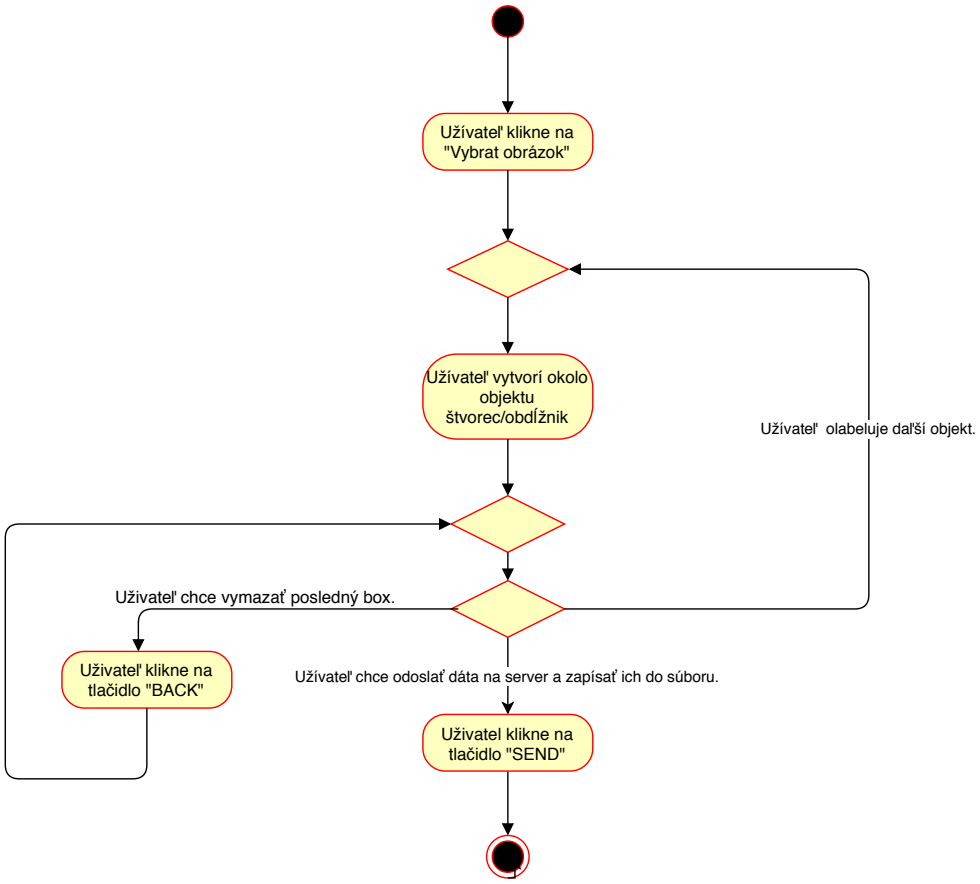
Stlačením tlačidla BACK sa užívateľ vráti o krok späť, teda posledný odĺžnik/štvorec sa odstráni.

Stlačením tlačidla SEND pošle označené súradnice na server, ktorý dáta uloží do súboru s názvom data.csv vo formáte:

$$< menozdroja > (< xmin, ymin, xmax, ymax >)^*.$$

Aplikácia sama doplní veľkosť štítku, aby odpovedala počtu najviac možných detekovaných osôb.

Server aplikácie je jednoduchý HTTP server napísaný v Node.js. Do budúcna je možnosť rozšíriť aplikáciu o funkcionalitu rozloženia videa na jednotlivé framy.



Obr. 5.2: Ukážka diagramu aktivít.

5.5 USC pedestrain dataset

Jeden z použitých datasetou pre detekciu osôb bol USC pedestrian set A/B [16] a USC pedestrian set C [17].

Dataset obsahuje sadu obrázkov vo formáte BMP a anotácie k nim vo formáte XML. Anotácie su uložené pod menom

$$< menoobrazka > .gt.xml.$$

Rootovým elementom v XML dokumente je element `<ObjectList>` ktorý obsahuje N elementov `<Object>`, kde N je počet osôb na obrázku. Každý `<Object>` element obsahuje element `<Rect>`, ktorý obsahuje attributy `height`, `width`, `x`, `y` popisujúce obdĺžnik udávajúci polohu osoby.Aplikácia načíta xml dokument pomocou python modulu `xml.etree.ElementTree`. Vyextraktuje attributy každého `<Rect>` elementu, vypočíta súradnice:

$$xmin = x,$$

$$ymin = y,$$

$$xmax = x + width,$$

$$ymax = y + height.$$

Súradnice všetkých osôb sú po tom uložené ako:

$$labels = [[box_1], [box_2]...[box_N]].$$

Kde N je počet osôb na obrázku. Do premennej *data* sa uloží pair (načítaný obrázok, štítky)

```
<?xml version="1.0"?>
 <ObjectList>
   <Object>
     <Rect x="2" y="17" width="33" height="81"/>
   </Object>
   <Object>
     <Rect x="41" y="18" width="33" height="77"/>
   </Object>
   <Object>
     <Rect x="78" y="21" width="31" height="77"/>
   </Object>
   <Object>
     <Rect x="113" y="21" width="30" height="74"/>
   </Object>
   <Object>
     <Rect x="142" y="22" width="28" height="73"/>
   </Object>
   <Object>
     <Rect x="169" y="21" width="31" height="73"/>
   </Object>
   <Object>
     <Rect x="208" y="17" width="28" height="77"/>
   </Object>
   <Object>
     <Rect x="234" y="17" width="28" height="76"/>
   </Object>
   <Object>
     <Rect x="263" y="17" width="27" height="76"/>
   </Object>
   <Object>
     <Rect x="294" y="14" width="28" height="78"/>
   </Object>
   <Object>
     <Rect x="326" y="19" width="26" height="73"/>
   </Object>
   <Object>
     <Rect x="361" y="30" width="25" height="62"/>
   </Object>
</ObjectList>
```

5.6 PennFudanPen dataset

Ďalším datasetom ktorý aplikácia využila bol PennFudanPen dataset [18]. Štítky k datasetu sú uložene v textovom dokumente.Prvých 8 riadkov dokumentu obsahuje informácie o obrázku.Od 9 riadku začína popis objektov, každý objekt je popísaný 5 riadkami (5ty reprezentuje prázdny riadok). Aplikácia vypočíta počet objektov a v cykle pomocou regulárnych várazov vyberie polohu obdĺžnika. Súradnice všetkých osôb sú po tom uložené ako:

$$labels = [[box_1], [box_2]...[box_N]]$$

.

Kde N je počet osôb na obrázku. Do premennej *data* sa uloží pair (načítaný obrázok, štítky).

```
1  # Compatible with PASCAL Annotation Version 1.00
2  Image filename : "PennFudanPed/PNGImages/FudanPed00001.png"
3  Image size (X x Y x C) : 559 x 536 x 3
4  Database : "The Penn-Fudan-Pedestrian Database"
5  Objects with ground truth : 2 { "PASpersonWalking" "PASpersonWalking" }
6  # Note there may be some objects not included in the ground truth list for
       they are severe-occluded
7  # or have very small size.
8  # Top left pixel co-ordinates : (1, 1)
9  # Details for pedestrian 1 ("PASpersonWalking")
10 Original label for object 1 "PASpersonWalking" : "PennFudanPed"
11 Bounding box for object 1 "PASpersonWalking" (Xmin, Ymin) - (Xmax, Ymax):
       (160, 182) - (302, 431)
12 Pixel mask for object 1 "PASpersonWalking" : "PennFudanPed/PedMasks/
       FudanPed00001_mask.png"
13
14 # Details for pedestrian 2 ("PASpersonWalking")
15 Original label for object 2 "PASpersonWalking" : "PennFudanPed"
16 Bounding box for object 2 "PASpersonWalking" (Xmin, Ymin) - (Xmax, Ymax) :
       (420, 171) - (535, 486)
17 Pixel mask for object 2 "PASpersonWalking" : "PennFudanPed/PedMasks/
       FudanPed00001_mask.png"
```

5.7 Architektúra siete

Keďže, detekcia osôb a nehumanoidných objektov nie je trivilána záležitosť, skonštruovaná architektúra siete je robustnejšia.Pri vytvárani bolo otestovaných viacero architektúr, každá obsahovala vyše 50 000 000 parametrov a nad 20 vrstiev.Výsledná architektúra, teda tá ktorá vykazovala najlepšie výsledky, a ktorej výsledky a experimenty na nej popisujeme, obsahuje 31 konvolučných vrstiev, za každou konvolučnou vrstvou nasleduje vrstva normalizačná. Táto sieť obsahuje 69 470 123 parametrov. Táto sieť vykazovala najlepšie výsledky pre detekovaní osôb. Pri detekovaní vozidiel boli použité daľšie 2 architektúry,

ktore vykazovali lepšie výsledky pri ich detekcií. Architektúra týchto sieti je popísaná v sekcii 5.8.4 Ako normalizačná funkcia bola vždy zvolená zvolená LeakyRelu [19]. Každá sieť prijíma na vstupe obrázok o veľkosti:

$$(N \times 416 \times 416 \times 3).$$

Kde N je po počet obrázkov.

Priechodom sieťou sa vstup upraví na $(N \times K)$. Kde N predstavuje maximálny počet osôb, ktoré vie sieť deketovať a K predstavuje atribúty, akými sú osoby reprezentované, teda informácie o polohe obdĺžniku, ktorým je osoba popísana. V prípade $K = 5$ prvá hodnota indikuje pravdepodobnosť, či sa niečo v obdĺžniku nachádza. Ostatné 4 hodnoty predstavujú $< xmin, ymin, xmax, ymax >$ kde:

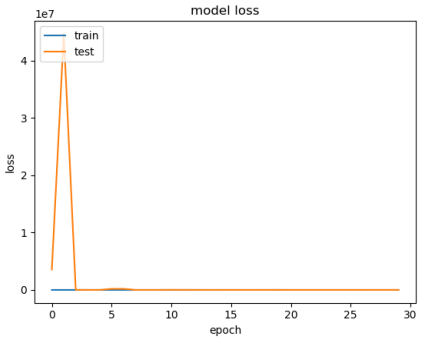
$(xmin, ymin)$ = poloha pravého horného bodu.

$(xmax, ymax)$ = poloha ľavého dolného bodu.

Jednotlivé obrázky architektúr sa nachádzajú v prílohe.

5.8 Trénovanie

Prvotné trénovanie prebiehalo na 360 vyhovujúcich obrázkoch z USC a PenFunDan datasetu. Veľkosť epochu, teda počtu opakovaní trénovania, bola nadstavená na 30. Batch size, teda počet obrázkov ktorý sa pošle cez sieť a následne sa váhy siete poupravia, bol nadstavený na 25, hlavne aj z dôvodu limitácie hardwaru. Testovanie prebehlo na 90 obrázkoch datasetu, ktoré pri učení sieť ani raz nevidela.



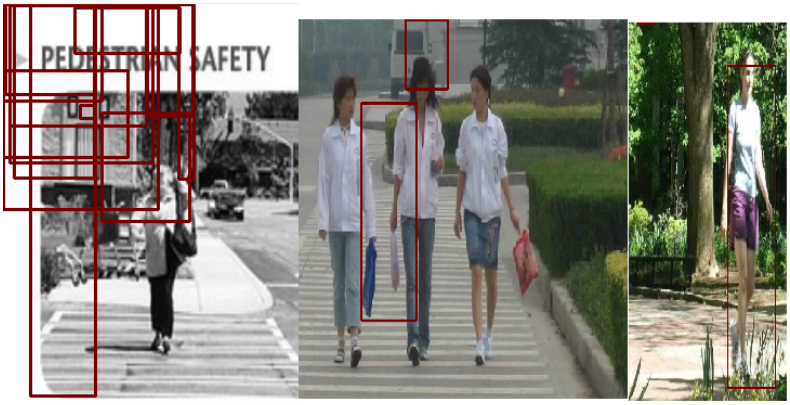
Obr. 5.3: Ukážka grafu straty pre prvotné trénovanie.

Na trénovaní vykazovala stratu 2505, pri testovaní vykazovala stratu 3800.

Sieť dokázala detekovať osoby na obrázku, avšak s veľkou presnosťou to bolo iba v prípade, že sa na obrázku nachádzala iba jediná osoba. V prípade, že sieť nedetekovala osobu presne, detekovala ju kúsok od jej aktuálnej pozície.

Obr. 5.4: Ukážka výsledku prvotného trénovania.

Problémové boli detekcie viac osôb, kde sieť detekovala viac ľudí ako jedného, nie všetkých, alebo úplne mimo. Všeobecne, čím viac osôb na obrázku je, tým väčšia pravdepodobnosť, že detekcia nebude presná.

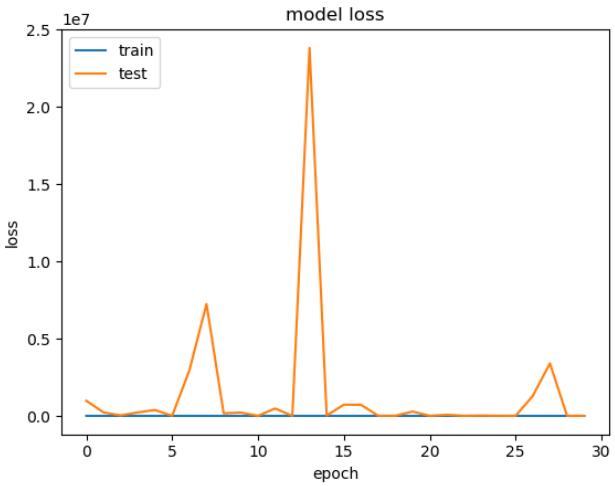


Obr. 5.5: ukážka problémových miest prvotného trénovania

5.8.1 Čiernobiele dáta

Jeden z možných vylepšení tréningu je upraviť dáta. Približne 25% dát použitých na trénovanie bolo farebných. Pretransformovanie týchto dát na čiernobiele môže viesť k vylepšeniu detekcie.Táto varianta dosiahla 92.5% úspešnosť pri trénovaní a stratu 3080.9996. Na testovacích dátach dosiahla 88.9% úspešnosť a stratu 4416.

Prevedenie farebných obrázkov nemalo takmer žiaden efekt na naučenie siete.



Obr. 5.6: Ukážka grafu straty pre trénovanie na čiernobielých obrázkoch.

Táto varianta nepriniesla veľké zlepšenie, ani veľkú stratu, no pri ďalších testoch budeme používať čiernobiele dáta, aby boli všetky dáta v rovnakom formáte.

5.8.2 Nápomoc pri trénovaní

Pri trénovaní sa porovnáva výsledok siete s požadovaným výsledkom,

$$[[output\_box1], [output\_box2], [output\_box3],$$

$$[[real\_box1], [real\_box2], [real\_box3]].$$

Poloha $output\_box_i$ sa porovná s polohou $real\_box_i$ Problém však nastáva, kedy $output\_box_i$ viacej odpovedá $real\_box_j$ ako $real\_box_i$. Sieť by mala teoreticky takýto problém vyriešiť sama, ale menšia pomoc má potenciál pomôcť sieti sa lepšie naučiť.

Tento problém sa dá vyriešiť prehadzovaním vstupných boxov. Po každom trénovaní otestujeme sieť na rovnakých dátach na ktorých sme ju trénovali a porovnáme jednotlivé vstupy a výstupy. Na základe tohoto porovnávania poprehadzujeme boxy[1] vo vstupoch aby boli na rovnakej pozícii ako ich najlepšia zhoda vo výstupe.

Môžeme to docieliť viacerými spôsobmi.

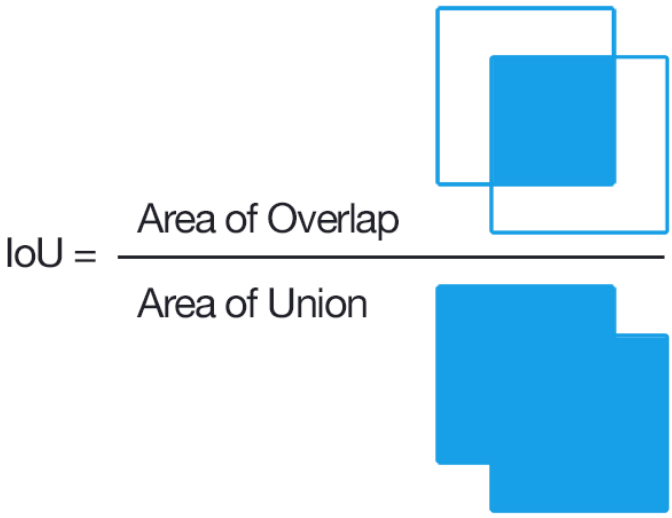
[1]Element predsatvujúci štítok

5.8.2.1 IOU - Intersection over Union

Jednou s možných porovnávacích metód, spomenuté o sekciu vyššie je IOU. Na jeho výpočet potrebujeme predikovanú polohu objektov ktorú vyhodnotí sieť, a skutočnú polohu.

IOU sa vypočíta ako:

$$IOU = \frac{plocha\ prieniku}{plocha\ zjednotenia}$$



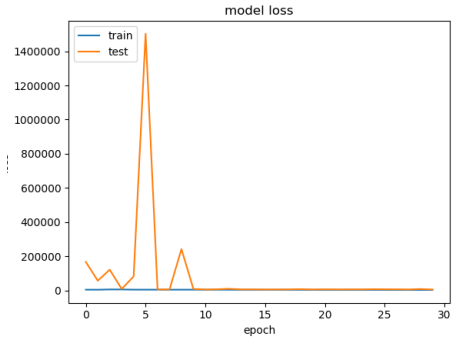
Obr. 5.7: Ukážka IOU.

Source: pyimagesearch.com

Pre každú kombináciu skutočnej a predikovanej polohy objektu vypočítame ich IOU, vyberieme najväčšiu hodnotu, a boxy ktorých IOU túto hodnotu predstavuje poprehadzujeme tak, že box reprezentujúci skutočnú polohu objektu dáme na pozíciu na boxu reprezentujúci predikovanú polohu a ich IOU nadstavíme na -1.

Ak box reprezentujúci reálnu polohu objektu je na vstupe na 3 pozícii, a box reprezentujúci predikovanú na 1 pozícii na výstupe, box reprezentujúci reálnu polohu premiestnime na 1 pozíciu na vstupe. Po vykonaní činnosti nadstavíme ich IOU na -1 a činnosť opakujeme N krát kde N reprezentuje počet predikovaných objektov.

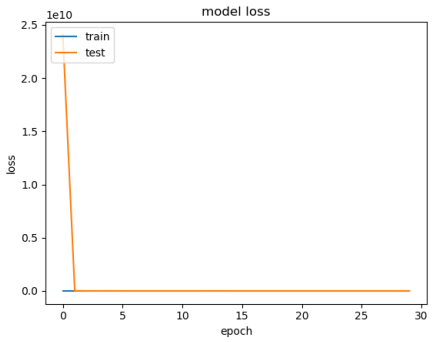
Pri tejto metóde bol dosiahla sieť najlepšiu stratu 2650 pri trénovaní, a stratu 5100 pri testovaní. Táto metóda nemala veľký vplyv na zväčšenie presnosti dát, ba naopak mierne negatívny vplyv. Potenciál tejto metódy nastáva pri zvýšení počtu dát. Nevýhodou je rýchlosť. Používaním tejto metódy sa sieť trénuje dlhšie.



Obr. 5.8: Ukážka grafu straty pri použití IOU.

5.8.2.2 MSE

Ďalšou alternatívou porovnávacej metódy je MSE. Rovnako ako pri IOU vypočítame MSE pre všetky kombinácie predikovaných a reálnych polôh objektov, vyberieme v tomto prípade najmenšiu hodnotu, prehodíme poradie boxov a ich MSE hodnotu nadstavíme na nekonečno.



Obr. 5.9: Ukážka grafu straty pri použitia MSE.

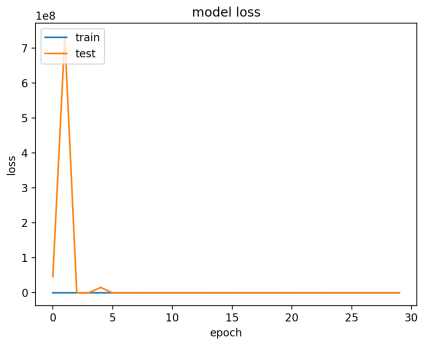
Najmenšiu stratu dosiahla táto metóda 4524 s presnosťou 89%. Pri testovacích dátach bola táto strata však v miliardách, teda bola extrémne neprestná. Pri vykonaní opätovných pokusoch táto metóda bola vždy extrémne nepresná, preto pri budúcich pokusoch budeme skúšať iba metódu IOU.

5.8.3 Experiment - obrázky s odstránením pozadím

Jedným zo zaujímavých experimentov, ktorý je možno vyskúšať, je pokúsiť sa neurónovú sieť naučiť detekovat objekty na upravených obrázkoch.

V tomto experimente bolo z čiernobielych obrázkov odstránené pozadie. Následne sa obrázky použili ako vstup do neurónovej siete. Táto varianta

skončila o niečo horšie ako pôvodné trénovanie siete.



Obr. 5.10: Ukážka grafu straty pri trénovaní na obrázkoch, z ktorých bolo odstránené pozadie.

Sieť vykazovala adekvátnu presnosť a stratu pri trénovaní aj testovaní, avšak taktiež mala problém s viacerými osobami. Pri použití natrénovanej siete na obrázky, z ktorých nebolo odstránené pozadie, sieť nevedela detekovať prakticky žiaden obrázok.

Obr. 5.11: Ukážka výsledku trénovania na odstránenom pozadí.

5.8.4 Nehumanoidné objekty

Na trénovanie nehumanoidných objektov, presnejšie áut, sa použil dataset oštítkovaný firmov crowdAI. Dataset pozostáva z vyše 9 000 obrázkov, získané z videa, ktoré sa natáčalo priechodom ulíc v Kalifornii z kapoty auta. Dataset síce obsahuje veľa obrázkov, ale väčšina obrázkov sú veľmi podobné, preto mal dataset pri každom trénovaní nábeh na preučenie aj za použitím výpadkovej vrstvy. Ďalšou možnou chybou v datasete je spôsob označovania objektov.

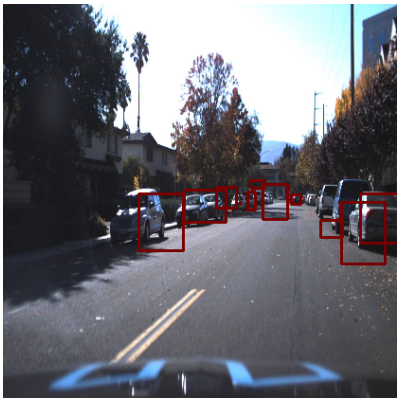
Na obrázkoch sú oštítkované všetky autá, aj tie, ktoré sú v diaľke a je ťažko rozoznateľné, či ide o auto alebo nie. Tieto autá, ktoré sú ťažko rozoznateľné, nedetekuje ani doposiaľ najlepší algoitmus YOLO.

Príklad detekcie YOLO algoritmu na datasete:

Obr. 5.12: Ukážka výsledku algoritmu yolo.

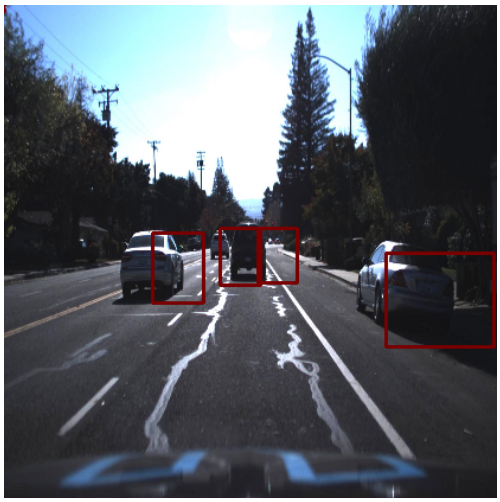
Ako je vidno, tento algoritmus nedetekoval vzdialené autá. Dôvod prečo tieto vzdialené auáo môžu predstavovať zlé dáta je spôsob tvaru aký predstavujú. Vo väčšine prípadov ide o farebný obdĺžnik, ktorý strieda 2 alebo 3 farby. Tento tvar môže nasimulovať aj diera v kríku alebo na ceste, na stene budovy.

Pre túto detekciu vykazovala najlepšie architektúra, ktorá ma obsahuje 65 641 796 parametrov. Ukážka detekcie natrénovanej siete:

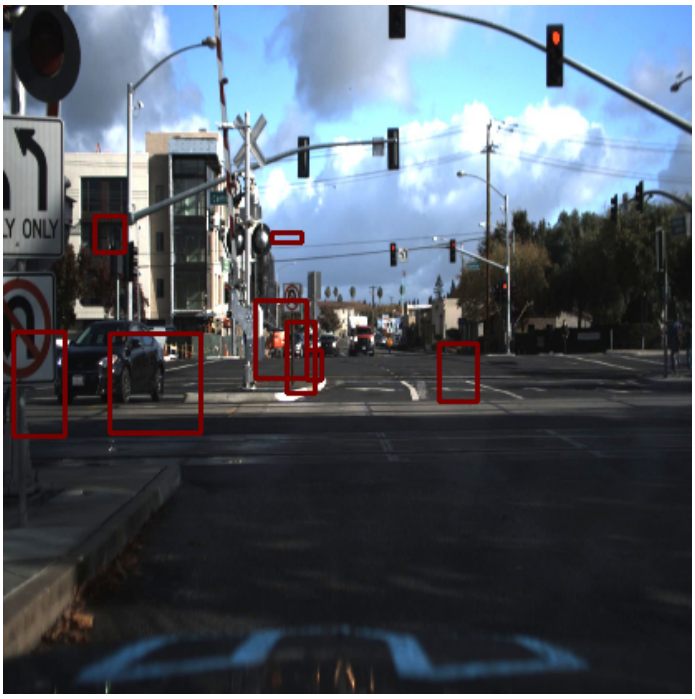


Obr. 5.13: Ukážka výsledku trénovania na autách.

Podobné dáta, a spôsob štítkovania spomenutý vyššie mal vplyv na trénovanie.



Obr. 5.14: Ukážka výsledku trénovania na autách.



Obr. 5.15: ukážka výsledku na obrázku z iného datasetu

Ukážka obrázku z iného datasetu, na ktorom sieť detekovala objekty.

Ako je vidno, sieť detekovala miesta kde nie je auto. Väčšina takých miest odpovedá problémovému štítkovaniu spomínaného vyššie. Pre ďalšie trénovanie sa do datasetu pridalo približne 3 000 obrázkov iného druhu z VOC

datasetu [22]. Trénovanie na pôvodnej sieti vykazovalo náhodné detekcie, avšak pri upravení sieti na 80 718 828 paremetrov trénovanie otočilo výsledok problému s preučením. Sieť sa naučila detekovať adekvátne autá z pridaných obrázkov a typovo podobné obrázky v prípade, že sa na ňom nachádzalo len jedno vozidlo, no skončila horšie pri detekcí viac vozidiel a typovo podobných obrázkov ako pôvodný dataset.

Ukážka detekcie vozidla:

Obr. 5.16: Ukážka detekcie na jednom aute.

Aj v tomto prípade sa prejavili chybné štítkovania. Sieť detekovala pre stredný obrázok tvar popisovaný výššie.

5.9 Tepelné mapy

Tvorba tepelných máp je triviálna záležítosť. Aplikácia najskôr vytvorí 2D pole, ktorého všetky hodnoty sú nadstavené na 0, preiteruje všetky boxy detekovaných objektov, a zvýši hodnotu poľa na takom indexe, ktoré obsahuje poloha objektu.

Ak teda aplikácia detekuje objekt na polohe $[0, 0, 100, 100]$, všetkým elementom v 2D poli, ktoré spĺňajú:

$$\sum_{i=xmin}^{xmax} \sum_{j=ymin}^{ymax} pole[i][j]$$

bude zväčšená hodnota.

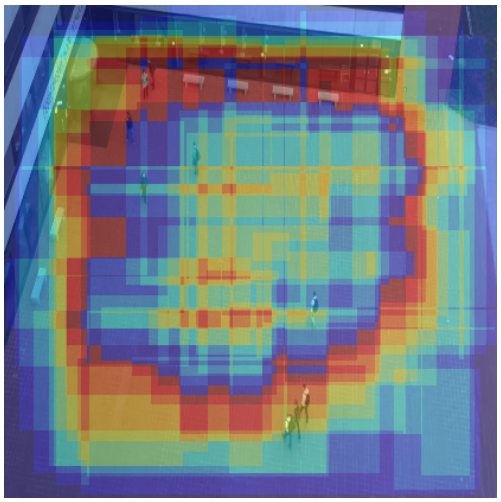
Podľa hodnoty sa danému elementu priradí farba a vytvorí sa tepelné mapa. Pre lepšiu prehladnosť sa tepelná mapa transparentne prekryje na pôvodný obrázok.

Ukážka farebnej škaly tepelnej mapy:

Obr. 5.17: Ukážka farebnej schémy.

Source: www.docs.opencv.org

Ukážka tepelnej mapy s náhodne vygenerovanými boxami:



Obr. 5.18: Ukážka tepelnej mapy.

Záver

Umelé neurónvé siete sú technológia závislá na dátach. Všeobecne, do určitej miery platí, čím väčší počet a väčšia kvalita dát, tým sa sieť naučí vykonávať svojú úlohu lepšie a presnejšie. Cieľom práce bolo detekovať objekty na videu/obrázku. Cieľ bol do určitej podoby splnený. Sieť vie detekovať osoby, avšak nie všetky a nie so 100% úspešnostou. Nehumanoidné objekty, v tomto prípade vozidlá vie sieť detekovať s určitou presnoťou a za určitých podmienok, teda ak sa na obrázku nachádza iba jedno vozidlo.

Veľkú príčínu na to majú hlavne dáta. Dáta použité na trénovanie neboli ideálne, a hlavne ich nebolo dostatok. Pre príklad, najlepšie siete boli trénované na 1 500 000 obrázkov, pričom daľšie dáta si vyrobili ich úpravou. Účelom nebolo deketovat špecifický druh nehumanoidného objektu alebo osoby, čo taktiež pridáva na obtiažnosti.

Dalšou príčinou je obmedzenia hardwaru a rýchlosť trénovania. Sieť sa v priemere pri 450 dátach trénovala 4 hodiny na grafike gtx 960m, čo nie je tak veľa, ale v prípade jemných úprav sa sieť musela natrénovať znova aby sa zistilo, či daná úprava mala pozitívny alebo negatívny vplyv. Pre trénovanie vozidiel bol na pár dní k dispozíci server s grafikou Tesla p100. Na tejto grafike sa 20 000 dát trénovalo 9 až 10 hodín.

V prípade náväznosti na túto prácu, bude treba obstarať kvalitné dáta vo veľkom počte, a obstarať prostredie pre jej trénovanie. Vybrané architektúry siete demonštrovali potenciál pri náznaku konvergencie k požadovanému výsledku, preto verím že pri dostatočnom výpočetnom výkone, a dostatočnom množstve dát, ktoré neobsahujú chýbne označenia, má sieť potenciál spĺňať svoj účel s malou chybovosťou.

Existuje pár metód, ktoré by zlepšili detekciu objektov v tejto práci, no ich použitím by aplikácia bola veľmi pomalá, čo je v tom prípade nežiadúce. Alternatívou je zobrať natrénovanú sieť a použíť tú, prípadne sa ju pokúsiť pretrénovať na požadovaný účel. Ďalšou alternatívou je zobrať obrázok miesta bez objektov, a následne pri každom frame detekovať, aké objekty sa nachádzajú na mieste pomocou odčítania pozadia, detekované objekty vložiť ako

vstup do neurónovej siete natrénovanú na klasifikáciu objektov, a v prípade že objekt patrí do typu ktorý chceme detekovať, vyznačiť ho na pôvodnom obrázku.

Literatúra

[1] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems.* 2012. p. 1097-1105.[online].[cit. 2018-05-05]. Dostupné z: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[2] HAN, Jun; MORAGA, Claudio. The influence of the sigmoid function parameters on the speed of backpropagation learning. In: *International Workshop on Artificial Neural Networks.* Springer, Berlin, Heidelberg, 1995. p. 195-201.[online].[cit. 2018-05-05]. Dostupné z: https://link.springer.com/10.1007%2F3-540-59497-3_175.

[3] BOJARSKI, Mariusz, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.[online].[cit. 2018-05-05]. Dostupné z: https://arxiv.org/pdf/1604.07316.pdf.

[4] (1999) *Collins Concise Dictionary, 4th edition,* HarperCollins, Glasgow, ISBN 0 00 472257 4, p. 1520.

[5] (1999) *Collins Concise Dictionary, 4th edition,* HarperCollins, Glasgow, ISBN 0 00 472257 4, p. 1386.

[6] (1999) *Collins Concise Dictionary, 4th edition,* HarperCollins, Glasgow, ISBN 0 00 472257 4, p. 328.

[7] LECUN, Yann, et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1989, 1.4: 541-551. [online].[ci. 2018-05-05]. Dostupné z: http://www.ics.uci.edu/~welling/teaching/273ASpring09/lecun-89e.pdf.

[8] ZEILER, Matthew D.; FERGUS, Rob. Visualizing and understanding convolutional networks. In: *European conference on computer vision.* Springer, Cham, 2014. p. 818-833.[online].[cit. 2018-05-05]. Dostupné z: https://arxiv.org/pdf/1311.2901.

[9] Hinton, G.E., Srivastave, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580,* 2012.[online].[cit. 2018-05-05]. Dostupné z: https://arxiv.org/pdf/1207.0580.pdf?utm_content=buffer3e047&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer.

[10] SCHERER, Dominik; MÜLLER, Andreas; BEHNKE, Sven. Evaluation of pooling operations in convolutional architectures for object recognition. In: *International conference on artificial neural networks.* Springer, Berlin, Heidelberg, 2010. p. 92-101.[online].[cit. 2018-05-05]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-642-15825-4_10.

[11] WANG, Zhou; BOVIK, Alan C. Mean squared error: Love it or leave it? A new look at signal fidelity measures. *IEEE signal processing magazine*, 2009, 26.1: 98-117.[online].[cit. 2018-05-05]. Dostupné z: http://ieeexplore.ieee.org/abstract/document/4775883/.

[12] *keras* [online].[cit. 2018-05-05]. Dostupné z: https://keras.io.

[13] *developer.nvidia* [online]. NVIDIA.[cit. 2018-05-05]. Dostupné z: https://developer.nvidia.com/cuda-zone.

[14] *developer.nvidia* [online]. NVIDIA.[cit. 2018-05-05]. Dostupné z: https://develcoper.nvidia.com/cudnn.

[15] PULLI, Kari, et al. Realtime computer vision with OpenCV. *Queue*, 2012, 10.4: 40.[online.][cit. 2018-05-05]. Dostupné z: http://lvelho.impa.br/ip08/reading/rt-ocv.pdf.

[16] WU, Bo; NEVATIA, Ramakant. Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In: *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on. IEEE*, 2005. p. 90-97.[online].[2018-05-05]. Dostupné z :http://ieeexplore.ieee.org/abstract/document/1541243/.

[17] WU, Bo; NEVATIA, Ram. Cluster boosted tree classifier for multi-view, multi-pose object detection. In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on. IEEE*, 2007. p. 1-8. [online]. Dostupné z: http://ieeexplore.ieee.org/abstract/document/4409006/.

[18] WANG, Liming, et al. Object detection combining recognition and segmentation. In: *Asian conference on computer vision. Springer, Berlin, Heidelberg*, 2007. p. 189-199. [online].[cit. 2018-05-05]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-540-76386-4_17.

[19] XU, Bing, et al. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015. [online].[cit. 2018-05-05]. Dostupné z: https://arxiv.org/abs/1505.00853.

[20] REDMON, Joseph, et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016. p. 779-788. [online].[cit. 2018-05-05]. Dostupné z: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf.

[21] *neuralnetworksanddeeplearning*[online]. Michael Nielsen. [cit. 2018-05-05]. Dostupné z:www.neuralnetworksanddeeplearning.com.

[22] EVERINGHAM, Mark, et al. The pascal visual object classes (voc) challenge. International journal of computer vision, 2010, 88.2: 303-338.[online].[cit. 2018-05-05]. Dostupné z: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

Dodatok A

Zoznam použitých skratiek

GUI Graphical user interface

XML Extensible markup language

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
- `imgs`
  - `network` ..... obrázky architektúr sieti
  - `testimgs` ..... testovacie obrázky