ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Lokalizace dronů v obrazu z kamery

Localization of UAVs from camera image

Diplomová práce

Studijní program: Kybernetika a robotika
Studijní obor: Letecké a kosmické systémy

Vedoucí práce: Mgr. RNDr. Petr Štěpán Ph.D.

Adam Ficenec

Praha 2016

FICENEC, Adam. *Lokalizace dronů v obrazu kamery*. Praha: ČVUT 2016. Diplomová práce. České vysoké učení technické v Praze, Fakulta elektrotechniky.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval (a) samostatně. Dále prohlašuji, že jsem všechny použité zdroje správně a úplně citoval (a) a uvádím je v přiloženém seznamu použité literatury.

Nemám závažný důvod proti zpřístupňování této závěrečné práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Praze dne: podpis: ……

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření

Akademický rok **2016-17**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	**Bc. Adam Ficenec**
Studijní program:	**Kybernetika a robotika**
Obor:	**Letecké a kosmické systémy**
Název tématu česky:	**Lokalizace dronů v obrazu z kamery**
Název tématu anglicky:	**Localization of UAVs from Camera**

Pokyny pro vypracování:

Seznamte se s metodami detekce obrazu pomocí neuronových sítí. Navrhněte neuronovou síť pro rozpoznávání dronů a tuto síť natrénujte. Napište aplikaci, která ve vstupním videu z kamery umístěné na dronu bude na výstup posílat souřadnice nalezených dronů v obraze. Otestujte spolehlivost algoritmu pro drony s umělým označením a bez přídavného označení.

Navrhněte algoritmus sledování dronů v obraze pomocí neuronových sítí.

Navržený algoritmus otestujte na jiných záznamech letu dronů a vyhodnoťte dosažené výsledky.

Seznam odborné literatury:

[1] A. Rozantsev, V. Lepetit and P. Fua. Detecting Flying Objects using a Single Moving Camera, accepted in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016.

[2] K. R. Sapkota, S. A. Roelofsen, A. Rozantsev, V. Lepetit and D. Gillet et al. Vision-Based Unmanned Aerial Vehicle Detection and Tracking for Sense and Avoid Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems, Daejeon, Korea, 2016.

Vedoucí diplomové práce:	Mgr. RNDr. Petr Štěpán, Ph.D. (K 13133)
Datum zadání diplomové práce:	29. září 2016
Platnost zadání do[1]:	28. února 2018

L. S.

Doc. Ing. Jan Holub, Ph.D.
vedoucí katedry

Prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 29. 9. 2016

[1] Platnost zadání je omezena na dobu tří následujících semestrů.

Poděkování

Děkuji především vedoucímu za možnost vypracování této diplomové práce a za jeho ochotu, rady a materiály bez kterých by nemohla vzniknout. Dále bych chtěl poděkovat své přítelkyni a rodině za podporu a pohodovou atmosféru, která mi velmi pomohla pro klidnou tvorbu celé práce.

Abstrakt

Tématem této práce je využití neuronových sítí pro lokalizaci kvadrokoptér. Řešení práce spočívá v důkladném prostudování látky a návrhu algoritmu, který bude schopen správně detekovat a lokalizovat UAV z kamerového výstupu pomocí neuronových sítí.

Práce se zabývá rozborem neuronových sítí, vhodným výběrem sítě pro řešení problematiky, jejím návrhem a vytvořením funkčního algoritmu schopného detekovat a označit objekty v reálném čase.

Kromě přípravy algoritmu pro živý vstup je jeho funkcionalita ozkoušena na testovací sadě obrazů pro získání ucelené informace o přesnosti. Výsledky testů jsou následně rozebrány a z nich jsou vyvozeny možné návrhy na zlepšení.

Klíčová slova

Neuronová síť, dron, detekce a lokalizace

Abstract

The goal of this work is to test the possibility of using neural networks to localize UAVs. Solution for this problem lies in an extensive research of given subject and in the development of algorithm, which will be able to detect and localize flying quadrocopters from video stream.

This work will provide a thorough analysis of neural networks, proper network design and the development of functional algorithm, capable of live stream object marking.

Apart from preparing the algorithm for live feed input, the functionality of this program will be tested on a set of pictures to properly analyze the precision. Results of these tests will be discussed in final suggestion for future upgrades.

Keywords

Neural network, UAV, detection and localization

Obsah

1 Úvod

Žijeme v době, kdy se výpočetní technika již nestává běžnou součástí našeho života, ale už se jím stala. Auta i letadla jsou z větší části ovládány elektronicky bez uživatelského zásahu, většina z nás vlastní inteligentní telefonní mobil a na světě je více počítačů než obyvatel planety, přičemž se toto číslo každým dnem zvyšuje. Jak roste naše potřeba a závislost na moderních technologiích, zvyšují se i naše nároky na její spolehlivost a schopnosti.

Rozvoj umělé inteligence je v tomto směru logickým a přirozeným vyústěním snahy lidí o automatizaci mnoha technologických odvětví. Přestože lidstvo ve velice krátké době udělalo neuvěřitelné pokroky na poli umělé inteligence, stále ještě je mnoho oblastí, kde se teprve během posledních let podařilo objevit či začít využívat potenciál v nich ukrytý. Jedním z těchto oblastí jsou i neuronové sítě.

Toto odvětví umožňuje strojům učit se na podobných principech jako lidský mozek, což vede ke schopnosti vykonávat složitější úkoly, než bylo dříve možné pomocí specializovaných algoritmů, popřípadě je zvládat mnohem efektivněji. Jejich potenciál je obrovský a možnosti využití zasahují od rozpoznávání, kryptografie, přes letectví až po odhadování vývoje burzovního trhu.

S tím, jak se rozvíjí tato nová odvětví a dochází k rychlé miniaturizaci elektronických zařízení, dochází i k přirozenému prolínání jednotlivých oborů, protože velikost a cena jednotlivých komponent umožňuje jejich plošné nasazení v nejrůznějších oblastech života.Jedním z takových případů je propojení umělé inteligence s letectvím, například v případě dronů, které jsou v dnešní době i při malé velikosti osazeny moderními senzory, mezi které patří například kamery. Díky tomu by mohli být brzy schopny vykonávat samostatně složité úkoly bez zásahu lidského faktoru.

Autonomní operativnost leteckých dronů je proto v dnešní době velice populární téma, kterému se věnují univerzity i armády po celém světě. Mezi důležité milníky v tomto oboru pak patří schopnost rozpoznávání objektů a jejich lokalizace v reálném čase. Tato meta je důležitá pro mnoho potenciálních úkolů, od mapování nedostupných terénů po navigaci roje dronů.

Cílem této práce pak je navrhnout neuronovou síť a naučit ji rozpoznávání objektů tak, aby byla schopná v reálném čase identifikovat a lokalizovat okolní drony pomocí své kamery a vyznačit ho v obraze. Práce se skládá z přehledové teoretické části, která se podrobně věnuje moderním neuronovým sítím, jejich tvorbou, architekturou a stavu problematiky metod pro identifikaci objektů v obraze.

Praktická část se zabývá vlastním návrhem neuronové sítě, včetně učícího postupu a metodou pro lokalizaci objektu v obraze včetně jeho realizace. Výstupem práce pak je program schopný rozpoznat a lokalizovat letící dron z živého přenosu kamery.

2 Neuronové sítě

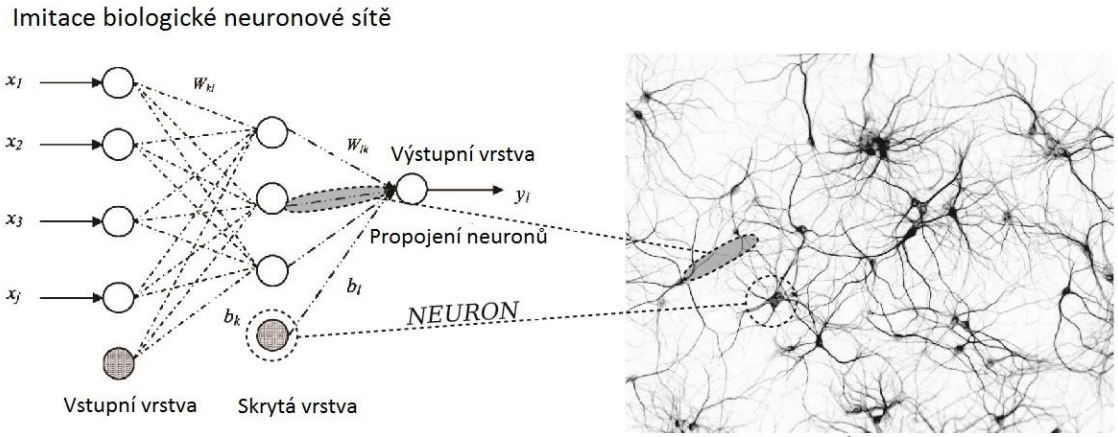
2.1 Úvod do umělé inteligence

Umělá inteligence je oblast vědy, zabývající se systémy, které jsou schopné napodobit inteligentní chování myslících organismů. Přesná definice je nejednoznačná, proto se většinou pouze porovnává s lidským rozumem, i když jde ve většině případů stále o velmi omezené schopnosti.

V dnešní době existuje mnoho odvětví, které dosahují vynikajících výsledků, které ještě před nedávnou dobou nebyly předpokládány. Mezi tato odvětví patří genetické programování, heuristické prohledávání stavového prostoru, data mining, strojové učení a z nich vycházející neuronové sítě.

2.2 Neuronové sítě

Jejich princip se objevuje již koncem 19. století, ovšem matematický vyjádřeny byly poprvé ž v roce 1949 matematikem jménem Donald O.Hebb. Základem neuronových sítí, jak již název napovídá, je napodobení činnosti lidského neuronu a jeho synapsí komunikujících s dalšími neurony pomocí receptorů. V lidském mozku podnět aktivuje neurony a ty přeposílají informaci dále do mozku, kde se celá operace opakuje. Tento systém je napodoben i v neuronových sítích.[1]

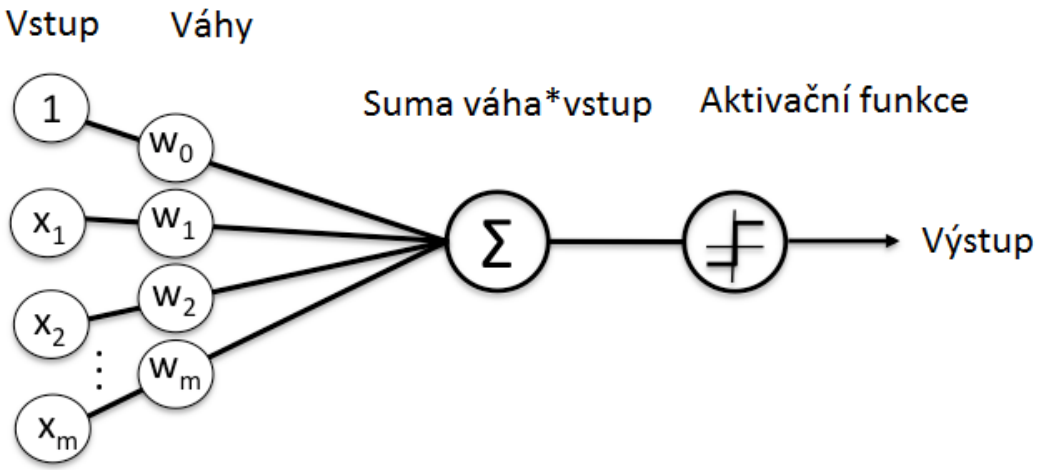


Obrázek 1-Princip imitace biologické sítě neuronů[1]

Základem celé struktury je model neuronu, který zpracuje jednoduché vstupy ve kterých hledá určité vzory, ty předá další sadě neuronů, která hledá vzorce ve vstupních vzorcích a tak dále. Těchto vrstev může být v řádu jednotek až desítek, podle architektury sítě a dostupného výpočetního výkonu. Vstupy jsou zpracovávány pomocí vah, které imitují různě silné synapse mezi jednotlivými neurony v mozku. Neuron sečte váhy všech vstupů, přičte vlastní důraz a vyhodnotí

[1] Obrázek převzat z http://rpi-cloudreassembly.transvercity.net/2012/11/04/neural-network-mapping-analysis-from-above/

výstup pomocí aktivační funkce. Výslednou hodnotu poté předá neuronům v dalších vrstvách. Neuron má libovolný počet vstupů, ale pouze jeden výstup. Nejpoužívanější model těchto neuronů je popsaný McCullochem a Pittsem a vypadá následovně:



Obrázek 2 - Základní princip umělého neuronu (10)

Hodnoty x_i vyjadřují vstup, velikost vah w_i vyjadřuje uložení zkušeností do neuronu. Čím je vyšší hodnota váhy w_i, tím je daný vstup důležitější. Tyto váhy jsou nastavovány ve fázi učení, které se budeme věnovat v další části. Aktivační funkce vyjadřuje způsob, jakým neuron na vstup reaguje. Podle typu neuronu a typu neuronové sítě se použije vhodná aktivace. Spojením libovolného množství těchto neuronů (libovolným způsobem) pak dostáváme neuronovou síť, kde poslední vrstvou je jednoduše počet neuronů odpovídající počtu potřebných výstupů (například pro klasifikaci 4 různých množin lze použít poslední vrstvu o 4 neuronech). Vrstvám, které se nacházejí mezi vstupní vrstvou a výstupní se říká skryté vrstvy (anglicky hidden layers).

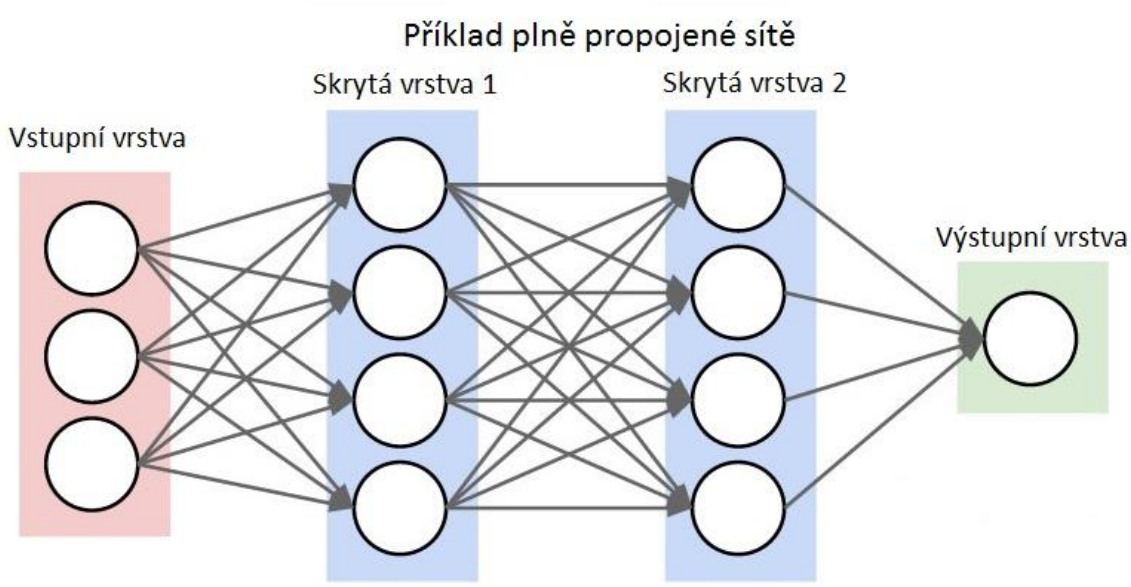
Neuronová síť nikdy nemůže fungovat sama od sebe. Stejně jako biologické neurony, i umělé neuronové sítě se nejprve potřebují "naučit" co vlastně mají detekovat/rozlišovat. Toto učení je esenciální, aby nám síť dávala přesné výsledky. V neuronových sítích se převážně používá metoda učení s učitelem, tedy porovnávání klasifikace se správnými výsledky a upravení hodnot vah vstupů na základě chyby. V případě neuronových sítí je struktuře předložen vzor, pro který je na základě aktuálního nastavení zjištěn výsledek. Ten je porovnán s požadovaným a je určena chyba (odchylka). Poté je spočítána korekce pomocí zpětné propagace výstupní odchylky a jsou upraveny hodnoty vah tak aby se snížila chyba. Proces se opakuje, dokud není dosaženo minimální chyby.

Tento postup si můžeme představit jako hledání minima v prostoru N-tého řádu a budeme se mu podrobně věnovat v další části. Náročnost této operace ovšem exponenciálně roste s velikostí

neuronové sítě, což byl také hlavní důvod, proč byly v minulosti neuronové sítě nahrazeny jednoduššími klasifikátory - běžné počítače neměli dostatečný výpočetní výkon pro zpracování vah v rozumném čase. Tento trend začal ustupovat až po přelomu tisíciletí s využitím paralelního zpracování dat, lehce dostupném s použitím výpočetního výkonu grafických karet systémem CUDA či OpenCL pro paralelní výpočet vah neuronové sítě. Klasická neuronová síť se tedy skládá ze tří hlavních částí: architektura (propojení neuronů) sítě, nastavení aktivační funkce a úpravy vah na základě zpětné vazby, tedy zpětné propagace (anglicky backpropagation). Jednotlivé části jsou popsány v následujících kapitolách.

2.2.1 Architektura neuronové sítě

Propojení neuronů v síti má silný vliv na funkčnost celé sítě. Nejpoužívanějším řešením je propojení, kde všechny neurony z jedné vrstvy jsou spojeny se všemi neurony z následující vrstvy. takzvaně plně propojená sít (anglicky fully connected network):



Obrázek 3 - Plně propojená neuronová síť (1)

Jednoduchost, stabilita a jednosměrné zpracování dat dělají z této architekty přirozeného kandidáta pro reálné aplikace neuronových sítí, protože pro daný vstup je vždy jasně definovaný výstup na základě naučených vah, a proto bude tato architektura použita i v této práci. Tato architektura může mít i různé varianty, kdy výstupy jedné vrstvy se rozštěpí a vedou do dvou a více dalších vrstev, aniž by tyto vrstvy již spolu reagovali. Plně propojená architektura je ovlivněna množstvím vrstev, jejich vzájemným napojením i různými hyperparametry, který mohou například dynamicky přetrhávat vazby mezi neurony během učícího procesu. Tyto parametry budou vysvětleny v následujících kapitolách.

Hopfieldowa cyklická síť je příkladem další architektury, kde jsou všechny neurony navzájem propojeny. Chování takové sítě je dynamické a více komplexní než u plně propojené, její využití pro praktické účely je však nízké a využívají se spíše pro optimalizaci propojení neuronů.

2.2.2 Aktivační funkce

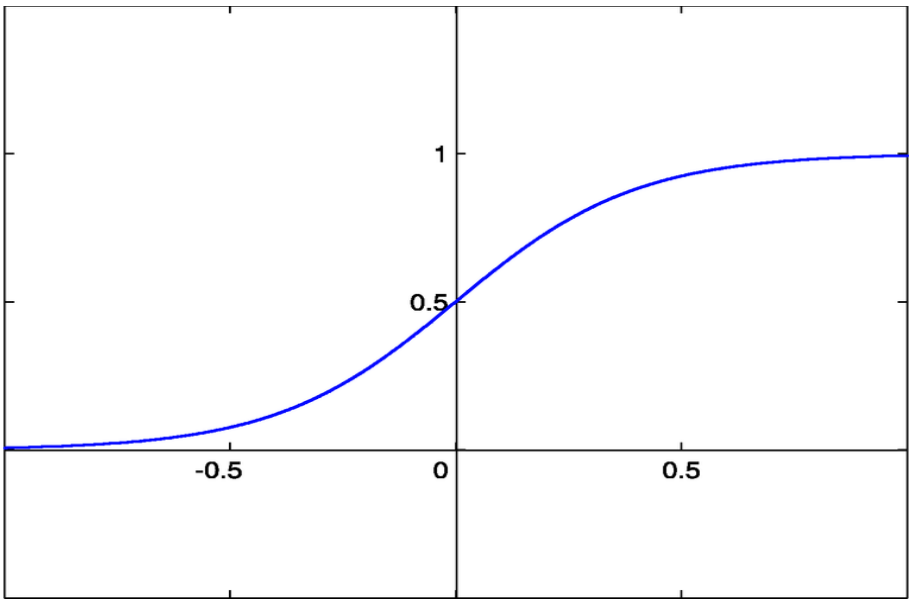
Zpracování vstupu a generování výstupu je klíčová část neuronu. V reálném světě je aktivační funkce neuronu reprezentována frekvencí, kterou neuron přeposílá informace dál do soustavy (naléhavost neuronu). Neuronové sítě tuto váhu reprezentují velikostí výstupní hodnoty, které se liší podle uplatněné funkce.

Nejjednodušší aktivační funkcí je jednoduchý binární spínač, využívající Heavisideovu funkci. V principu se neuron aktivuje při překročení určité hodnoty ze vstupních vah a na výstup pošle 1, v opačném případě 0. Tato funkce byla dlouhou dobu základem neuronových sítí, která ovšem zpomalovala jejich rozvoj, protože kvůli její jednoduchosti je potřeba velké množství neuronů pro zpracování signálu. Hodí se pro binární klasifikace.

Další, dnes velmi rozšířenou aktivační funkcí je spojitá funkce:

$$S(a) = \frac{1}{1 + e^{-a}} \tag{1}$$

Obecně se tato funkce nazývá funkce sigmoid a graf této funkce je následující:

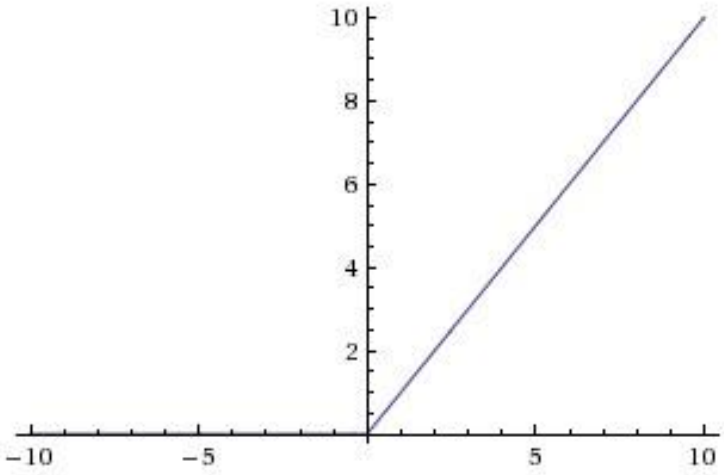


Obrázek 4 - Průběh funkce sigmoid

Z obrázku je patrné, že výstup neuronu se blíží logické 1 anebo 0, podle součtu vstupních vah. Zjednodušeně by se dalo říct, že tato hodnota vyjadřuje jistotu, s jakou byl daný podnět neuronu správně vyhodnocen. Výhodou této funkce je (kromě její podobnosti s biologickým

zpracováním podnětu v neuronu) jednoduchost výpočtu derivace. To je praktické při nastavování vah v učící fázi, které je řešeno pomocí gradientů chybovosti a bude vysvětleno v následující kapitole.

Třetí nejpoužívanější funkcí je tzv. ReLu funkce (z anglického rectified linear unit). Tato funkce jednoduše saturuje všechny hodnoty menší než 0 na 0. Tedy **R(z) = max(0,z).**



Obrázek 5 - Průběh funkce ReLu

Tato funkce poslední dobou nabírá na popularitě. Mezi hlavní výhody patří mnohem vyšší rychlost při nastavování vah v učící fázi a její jednoduchost (1). Hlavní nevýhodou je situace, kdy tok gradientu je příliš rychlý, což způsobí nastavení nízkých hodnot vah blízkou nule, která vede k nepoužívání ("smrti") neuronu, který se kvůli nízkým hodnotám vah nikdy neaktivuje. Tento jev je možné korigovat nastavováním dalších obecných parametrů sítě, kterým se říká hyperparametry (a které budou vysvětleny v následujících kapitolách).

Poslední rozšířenou aktivační funkcí, která se používá pro finální klasifikaci je tzv. funkce softmax. Tato funkce normalizuje výstupy tím způsobem, že zobrazí vektor $\vec{v}$ **(1,n)** výstupních hodnot do vektoru $\vec{s}$**(v)** stejné velikosti ovšem s hodnotami v rozmezí **(0,1)** tak, že suma všech hodnot je 1. Tím je vyjádřena pravděpodobnost, s jakou je daná klasifikace správná, nehledě na původní, nenormalizovaný výstup. Rovnice této funkce je:

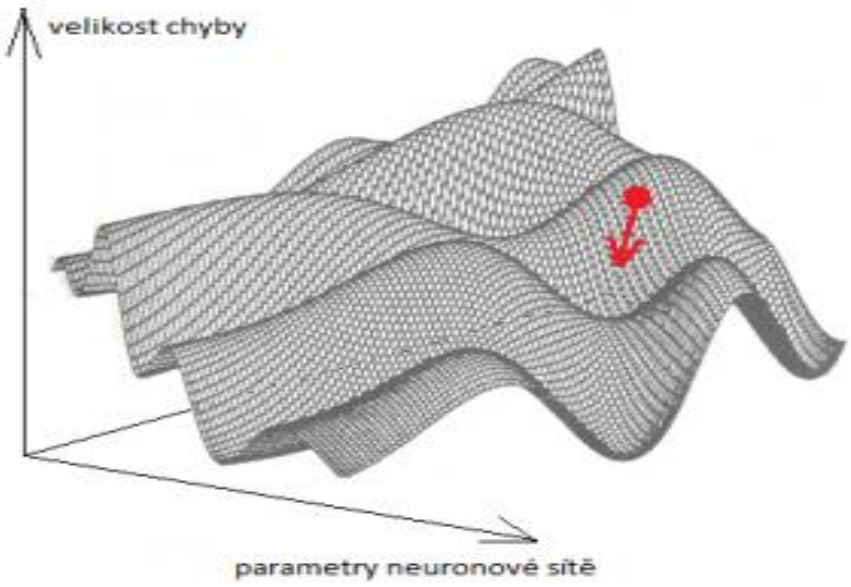
$$\vec{s}(\mathbf{v})_j = \frac{e^{v_j}}{\sum_{n=1}^{n} e^{v_n}}, \; j = 1,\ldots,n \tag{2}$$

Tato funkce bývá výstupem sítě a při porovnávání s odchylkou se jednoduše porovná vzdálenost očekávané klasifikace x_i od 1, tedy:

$$1 - x_i \tag{3}$$

2.2.3 Váhy neuronů - zpětná propagace

Zpětná propagace je algoritmus pro nastavování vah spojení jednotlivých neuronů a je klíčovým faktorem funkcionality celé sítě. Algoritmus zpětné propagace se skládá z několika částí. Jak bylo zmíněno v předešlé kapitole, v principu jde o hledání minima, tedy hodnoty vah, při které se odchylka výstupu blíží nule. Pokud si představíme následující graf, kde svislá osa znázorňuje odchylku výstupu sítě od správné hodnoty a zbylé osy jako parametry sítě, pak pro různé zvolené parametry vah je přesně určená odchylka od očekávané hodnoty. Cílem je nalézt nejnižší "údolí" představující minimální chybu výstupu sítě:



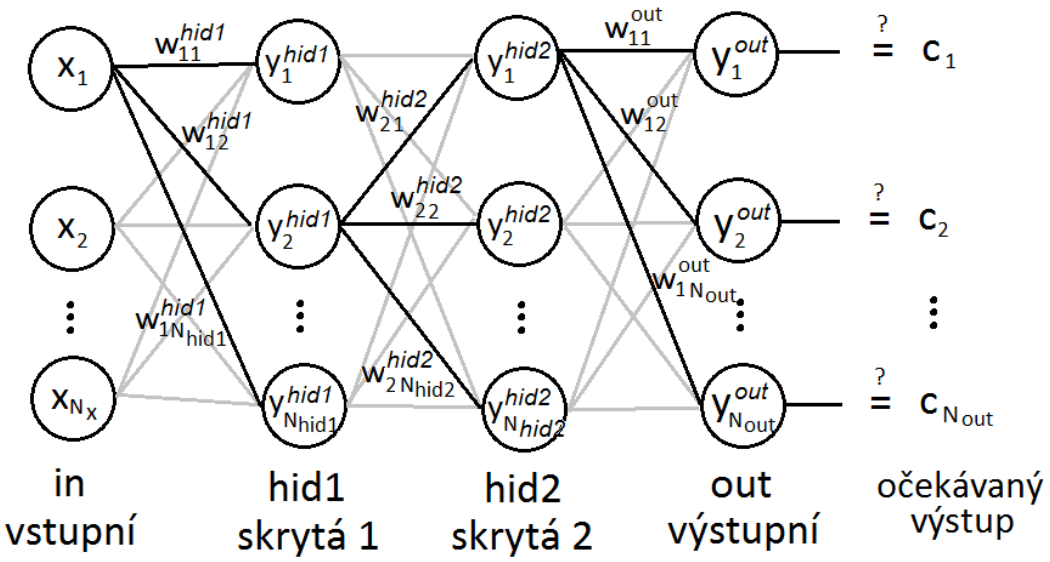
Obrázek 6 - Princip hledání minimální chyby (11)

Nejprve jsou náhodně určeny parametry sítě (zvolení náhodného bodu grafu v předchozím obrázku). Poté vezmeme vzorek z trénovací sady u kterého je známy správný výsledek a vyhodnotíme výstup sítě. Výstupní odchylka sítě od skutečné hodnoty je zpětně použita jako vstup a síť odzadu počítá vektor rychlosti růstu chyby (gradient) jednotlivých vrstev sítě a tyto hodnoty poté upravuje pro každou váhu. Proto název zpětná propagace.

Pro úpravu vah $W_{i,j}$ dané vrstvy se tedy snažíme najít gradient popisující změnu odchylky s vahou:

$$\frac{\partial chyba}{\partial W_{ij}^{vrstva}} \rightarrow \frac{\partial E}{\partial W_{ij}^{vrstva}} \tag{4}$$

Výpočet gradientu dané neuronové vrstvy se řídí řetízkovým pravidlem pro parciální derivace, kdy se musí nejprve spočítat gradient všech předchozích vrstev, počínaje výstupní vrstvou. Protože je toto téma poměrně náročné na detailní popsání, bude tento výpočet rozdělen do 4 dílčích výpočtů, která se skládají dohromady v závislosti na velikosti a architektuře sítě - pro příklad bude síť vypadat následovně:



Obrázek 7 - Příklad sítě pro zpětnou propagaci (11)

Prvním krokem je výpočet derivace chybové funkce (opět kvůli řetízkovému pravidlu pro parciální derivace). Pokud jako ztrátovou funkci pro daný výstup použijeme kvadratickou odchylku dostáváme:

$$Ec_i = \frac{d}{dx}\left(\frac{1}{2}(x_i - a_i)^2\right) = c_i - y_i^{out} \tag{5}$$

kde *y* je výstup a *c* značí očekávaný výstup.

Druhým krokem je výpočet derivace aktivační funkce. Pokud použijeme aktivační funkci sigmoid je derivace velmi jednoduchá (což je právě velkou výhodou této aktivační funkce):

$$S(y) = \frac{1}{1+e^{-y}} \rightarrow \frac{dS}{dy_i} = y_i^{out}\left(1 - y_i^{out}\right) \tag{6}$$

Spojením těchto kroků dostáváme vzorec, jak rychle se mění chyba po součtu všech vstupů v aktivační funkci - EI_i, tedy:

$$EI_i^{out} = Ec_i y_i^{out}\left(1 - y_i^{out}\right) = y_i^{out}\left(1 - y_i^{out}\right) = \left(c_i - y_i^{out}\right) \tag{7}$$

Třetím krokem je výpočet rychlosti s jakou se mění chyba po vynásobením konkrétní váhy -EW_{ij} . Vzhledem k tomu, že váhy jsou pouhým multiplikátorem je tato hodnota přímo úměrná jejich velikosti, dostáváme tak úpravu vah pro poslední vrstvu sítě:

$$\frac{\partial E}{\partial W_{ij}^{out}} = EI_j y_i^{hid2} = y_i^{hid2} * (y_i^{out}(1 - y_i^{out}) * (c_i - y_i^{out}) \quad (8)$$

Poslední krokem je výpočet, jak rychle se mění odchylka se změnou vah v předchozí skryté vrstvě - EV_i. Tento krok je esenciální pro mnohovrstvé sítě a umožňuje výpočet vah libovolného množství vrstev. Změna vah v předchozí vrstvě ovlivňuje odchylku ve všech uzlech, na které je daná vrstva přímo napojena, protože ve vrstvě bývá několik neuronů. Jedná se tedy o sumu dílčích výpočtů *j* z druhého kroku vynásobeným váhou W_{ij} vedoucí z daného uzlu:

$$EV_i = \sum_j EI_j^{out} W_{ij}^{out} \quad (9)$$

Kombinací těchto 4 kroků a řetízkového pravidla dostáváme rychlost změny odchylky pro libovolnou vrstvu a váhu ΔW_{ij}^{vrstva} . Například pro vrstvu hid2, která je opět tvořena aktivační funkcí sigmoid, přidáme sumační komponent a komponent derivace aktivační funkce:

$$\frac{\partial E}{\partial W_{ij}^{hid2}} = y_i^{hid1} * y_i^{hid2}\left(1 - y_i^{hid2}\right) * \sum_j EI_j^{out} W_{ij}^{out} \quad (10)$$

Kde $\sum_j EI_j^{out} W_{ij}^{out}$ je hodnota známá z předchozího kroku. Posledním krokem je pak samotná úprava váhy W_{ij} ve *vrstvě*:

$$W_{ij}^{vrstva} = W_{ij}^{vrstva} - \eta \frac{\partial E}{\partial W_{ij}^{vrstva}} \quad (11)$$

kde η představuje tzv. hyperparametr sítě, v tomto případě jde o rychlost učení (angl. learning rate). Pokud je tento parametr příliš vysoký, bude hledání minima oscilovat, naopak pokud je nízký, zpomaluje se doba učení. Hyperparametrům je věnována jedna z následujících kapitol. Celý postup "učení" pro neuronovou síť je tedy následující:

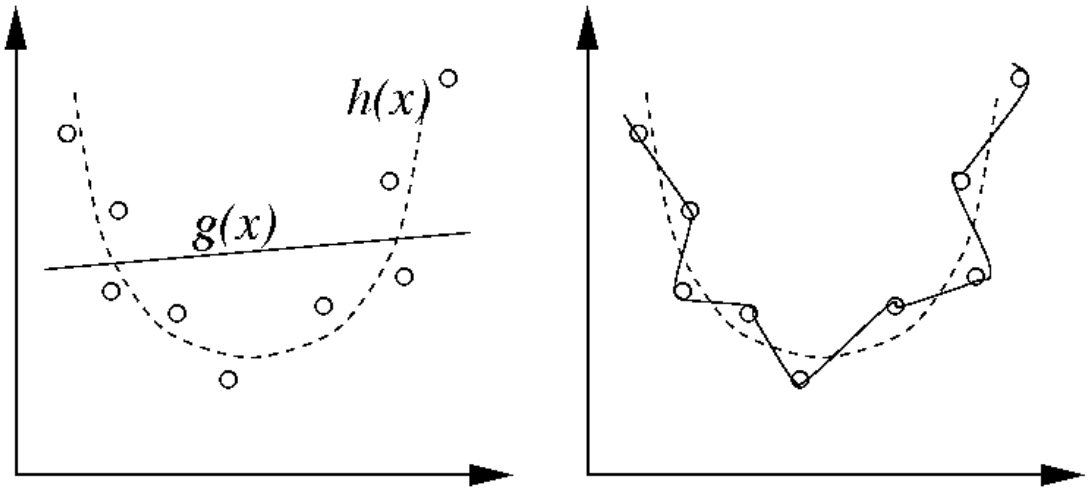
1. Načtení vstupu a vyhodnocení výstupu na základě aktuálního nastavení vah neuronové sítě

2. Porovnání výstupů sítě se skutečnou hodnotou a výpočet odchylek
3. Zpětná propagace těchto odchylek postupně pro všechny váhy sítě a výpočet gradientu pro každou z nich
4. Aktualizace vah
5. Opakování od 1 pro další vstup (obraz)

Tento cyklus se opakuje, dokud se nedosáhne požadovaného minima nebo dokud síť správně neohodnotí všechny učící vzorky, popřípadě předem určený počet vzorků. Je důležité zmínit, že síť se může "učit" z konkrétního vstupu i vícekrát. Ve skutečnosti je tento případ velice žádaný a pro nalezení vhodného minima jsou sítě nastaveny, aby procházeli celou učící množinu i mnohokrát za sebou. Pro projití celé množiny se používá název epocha. Reálné systémy velmi komplexních sítí pak během učícího procesu projdou i desítky epoch než začnou vykazovat ucházející výsledky.

2.2.4 Přeučení sítě

Cílem zpětné propagace je hledání globálního minima. Ovšem ne vždy, spíš výjimečně, je toto minimum vhodné při změně vstupních dat. Pokud síť necháme dostatečně dlouho učit na konečné množině dat, může se stát, že hodnoty vah se přizpůsobí konkrétním vstupním datům a nebudou již umět generalizovat vlastnosti detekovaných objektů, viz obrázek:



Obrázek 8 - Příklad optimalizované a přeučené sítě (15)

Nalevo vidíme dvě funkce vstupů, které vyjadřují nastavení vah. Funkce g(x) představuje síť, která nebyla naučena na množině zobrazených vstupů, popřípadě nebyla nastavena vůbec nebo je tvořena málo parametry. Tato síť je příliš jednoduchá a intuitivně můžeme říct, že by její klasifikace ve většině případů nebyla správná. Funkce h(x) představuje ideálně naučenou síť, která správně na základě vlastností vzorků bude schopná odhadnout i klasifikaci nových vzorků - zpětná propagace zde dosáhla optima. Napravo je funkce p(x), která byla přeučena. Problémem takového

přeučení bývá velké množství vah, doba učení, nebo malé množství učících vzorků. Síť však vykazuje výborné výsledky s minimální chybou pro danou množinu vzorků a na první pohled se může jevit jako dobré řešení. Pokud bychom se ovšem snažili vyhodnotit nový vzorek, je velká šance, že by předpověď selhala. Řešením tohoto problému je velká učící množina obsahující co nejvíce poloh detekovaného předmětu, úprava dat (například použití zrcadlových obrazů pro klasifikaci objektů) a správné nastavení hyperparametrů sítě.

2.2.5 Neuronové sítě - shrnutí

Klasické neuronové sítě jsou moderní a efektivní metoda pro klasifikaci různých druhů signálů. Cílem této práce je ovšem klasifikace celých obrázků, což je pro klasické neuronové sítě obtížné. Pokud si představíme jednoduchý černobílý obrázek jako mřížku tvořenou jednotlivými pixely v černé a bílé a pokusili bychom se je použít jako vstup pro neuronovou síť, i malá změna polohy obrázku by znamenala velmi rozdílné vstupní parametry pro neuronovou síť a tím i velmi rozdílné výstupy. Jinými slovy neuronová síť je velmi náchylná k deformaci vstupních parametrů. Proto musíme nějakým způsobem zajistit, aby byla neuronová síť dostatečně robustní vůči takovým posunům. Tímto problémem se zabývají právě konvoluční sítě (v angličtině convolutional neural networks, dále CNN).

2.3 Konvoluční neuronové sítě

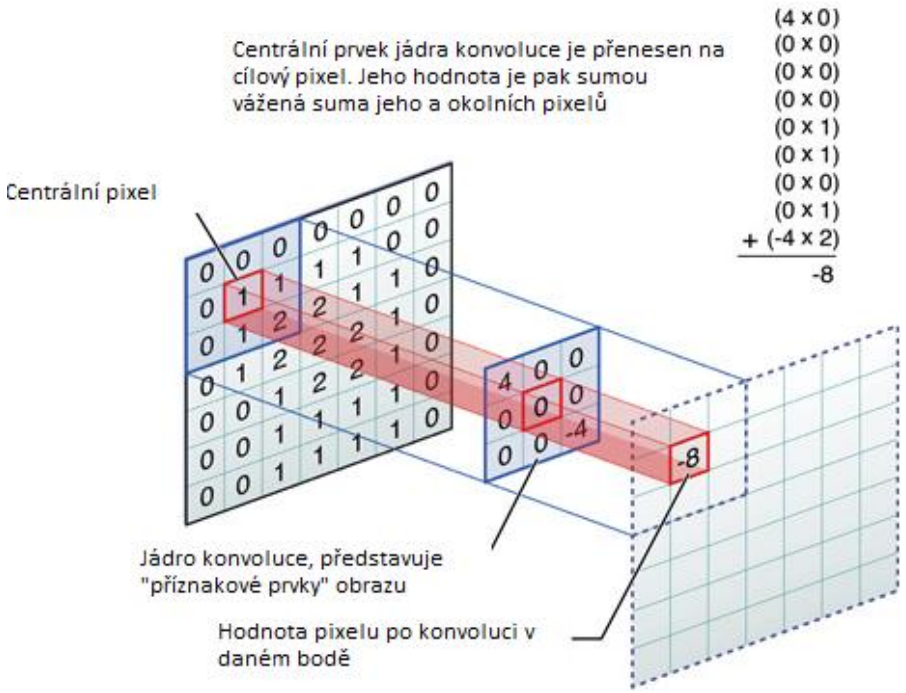
Základem konvoluční neuronové sítě je uplatnění konvoluce na vstupní data, tedy nejčastěji na vstupní obraz. CNN je rozšíření klasické neuronové sítě a sdílí mnoho jejích principu a z velké části je tvořena standardní neuronovou sítí. Konvoluční neuronová síť je definována váhami, strukturou propojení neuronů a používá mnoho shodných hyperparametrů (jako například learning rate). Každá CNN se skládá z několika vrstev. Tyto vrstvy se rozdělují na konvoluční, podvzorkovací (tzv pooling) a plně propojené vrstvy, což jsou klasické neuronové sítě, které byly popsány v předchozí kapitole. Důležitou součástí detekce objektů konvoluční sítí je i úprava obrazu před vstupem do sítě.

2.3.1 Počáteční zpracování obrazu

Přestože konvoluční část sítě může teoreticky zpracovat objekt libovolné velikosti (konvoluce a pooling nejsou závislé na velikosti vstupu), musí mít výstup konvoluční části pevnou velikost kvůli napojení na plně propojenou síť. Obraz je proto před konvoluční vrstvou transformován na požadovanou velikost, případně ještě převeden do černobílého obrazu. Tyto parametry jsou nastaveny před učící fází a nemění se (změna hodnot by vedla k rozbití naučených vah).

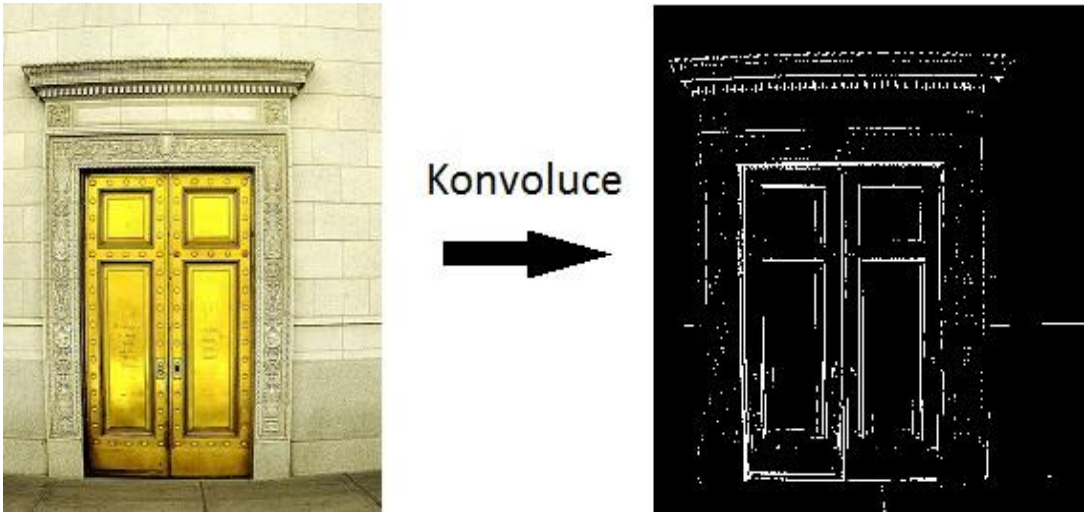
2.3.2 Konvoluční vrstva

Konvoluční vrstva na rozdíl od klasické vrstvy řadí váhy neuronů do 3D obrazců, které lépe pracují se vstupními obrazy. Tyto váhy jsou tak uskupeny do určité výšky, šířky a hloubky podle počtu vah a říká se jim konvoluční mapy. Šířka a výška je definována počtem vah v dané ose a v podstatě vyjadřuje počet pixelů, které daná mapa zpracuje. Hloubka definuje, v kolika základních barvách bude obraz zpracován. Protože klasický barevný obrázek je složen z různé intenzity modré/červené/zelené barvy, je tato hloubka pro barevné obrázky velikosti 3, pro černobílé pak 1 (odstíny šedi). Pro každou hloubku existuje vlastní sada vah, která zpracovává vstupy v odstínech dané barvy. Pro barevné obrázky se tak ztrojnásobuje počet parametrů vah. Konvoluční mapa je několikanásobně menší než samotný obraz, zpravidla má velikost několika pixelů (např. 3×3, 5×5...) ovšem není to podmínkou. Samotná konvoluce pak zpracovává část vstupního obrazu a danou konvoluční mapu neboli jádro. Jádro postupně prochází obraz definovaným krokem a vytváří konvoluci z těchto vstupů. Definovaný krok (anglicky stride)



Obrázek 9 - Princip konvoluce obrázku (14)

určuje, o kolik pixelů bude konvoluční mapa v obraze posunuta před každou konvolucí. Vyšší hodnota urychluje výpočet a snižuje velikost výstupu za cenu snížení přesnosti. Konvoluce spočívá v pronásobení překrývajících se pixelů dané části obrazu a mapy a sečtení těchto hodnot pro výsledné skóre, které vyjadřuje shodu dané části obrazu s danou mapou - čím vyšší skóre, tím vyšší podobnost. Pokud si představíme, že daná konvoluční vrstva má nastavené váhy tak, že připomínají svislou čáru, bude zvýrazňovat všechny oblasti obrazu, které mají podobnou charakteristiku. Tyto oblasti jsou pak na výstupu zvýrazněny:



Obrázek 10 - Příklad zvýraznění hran pomocí konvoluce

Myšlenkou konvoluční vrstvy je získání lokální charakteristiky obrázku. Tyto charakteristiky si můžeme (a síť takové skutečně buduje) představit jako různé výrazné hrany v obrazu (viz obr. č. 10). Pomocí konvolučních map je obraz přetvořen tak, aby byly zvýrazněny potřebné příznaky, které se síť nejprve sama musí naučit. Protože každá konvoluční mapa zpracovává pouze určitý rys, nebo vlastnost obrazu, musí pro každý rys existovat vlastní mapa/jádro. Čím více jader použijeme, tím více rysů se může síť naučit, ale hrozí zde možnost přeučení sítě.

Konvoluce snižuje velikost obrazu, což je žádoucí vlastnost, protože zrychluje zpracování rozměrných obrazů. Pokud je výška a šířka původního obrazu $\boldsymbol{w}$ a $\boldsymbol{h}$, výška/šířka jádra $\boldsymbol{a}$ a $\boldsymbol{b}$, pak pro výslednou výšku/šířku $\boldsymbol{w'}$ a $\boldsymbol{h'}$ a velikost definovaného kroku $\boldsymbol{S}\times\boldsymbol{S}$, pak platí:

$$\boldsymbol{w'} \times \boldsymbol{h'} = ((\boldsymbol{w}-\boldsymbol{a})/\boldsymbol{s}+\boldsymbol{1}) \times ((\boldsymbol{h}-\boldsymbol{b})/\boldsymbol{s}+\boldsymbol{1}) \qquad (12)$$

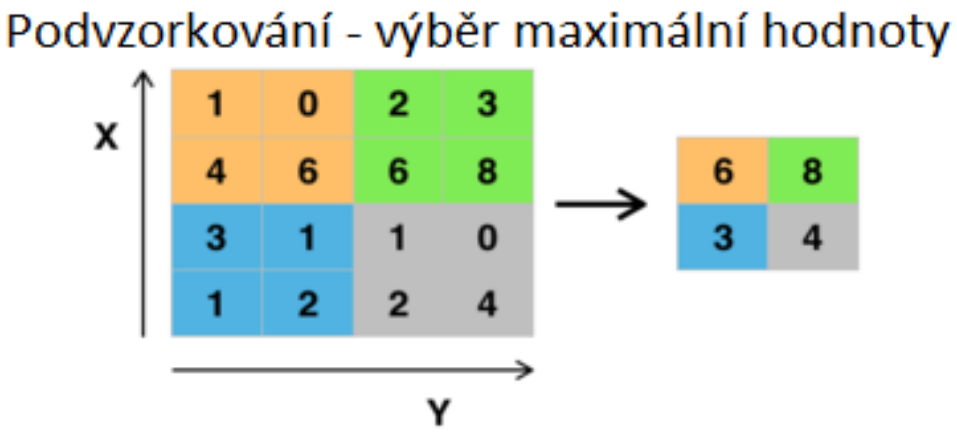
Jedna vrstva konvoluce by vedla pouze k malému zmenšení obrazu. Konvolučních vrstev proto bývá několik za sebou s proměnlivou velikostí. Principem je, že mapy fungují jako filtry pro různé zajímavé charakteristiky. Pokud poskládáme několik takových filtrů za sebe, bude síť na základě zpětné propagace ze zpracovaných výstupů hledat stále složitější a komplexnější struktury. V posledních vrstvách to mohou být například rysy obličeje, tvary auta atd.

Při použití malých konvolučních map by trvalo příliš dlouho, než by se obraz zmenšil natolik, aby byl vhodný pro vstup do plně propojené vrstvy sítě. Proto je mezi každé konvoluční vrstvy filtrování vložena vrstva podvzorkování.

2.3.3 Podvzorkovací vrstva

Funkce podvzorkovací vrstvy je čistě snížení velikosti vstupního obrázku a tím pádem i učících parametrů. Základním principem je výběr vhodné hodnoty z několika sousedních pixelů v obraze. Nejstarším a nejjednodušším typem podvzorkování je zprůměrování sousedních pixelů obrazu. Přestože byl výběr průměrné hodnoty dříve často používán, zjistilo se, že pro neuronové sítě je nejvhodnější vybrat největší hodnotu (anglicky max pooling).Maximální hodnota totiž bude

zvýrazňovat zajímavé body v obraze, což vede k výraznějším aktivacím (2). Velikost podvzorkování bývá zpravidla 2×2 s krokem 2, to znamená, že funkce se bude aplikovat na pixely o mřížce 2×2 a poté se posune o dva pixely na následující čtveřici. Další používanou variantou je uplatnění podvzorkování na mřížku o velikosti 3×3 a kroku 2, tzn překrývající se podvzorkování. Další varianty (větší mřížka) se nepoužívají, protože jsou příliš destruktivní a při jejich aplikaci se ztrácí informace důležitá pro rozpoznání správné klasifikace. I v případě mřížky 2×2 dochází ke ztrátě informace u 75% pixelů - ze 4 pixelů je vybrána pouze 1 hodnota, viz obrázek:



Obrázek 11 - Podvzorkování obrazu (3)

Pro zpětnou propagaci je nutné si u vrstvy podvzorkování pamatovat polohu pixelu, který je zdrojem výsledné hodnoty, aby se zpětně detekovaná chyba nepropagovala přes všechny pixely, které nepřispěly k podvzorkování. Toto pamatování polohy přispívajícího pixelu zajišťuje úspornější hledání minima.

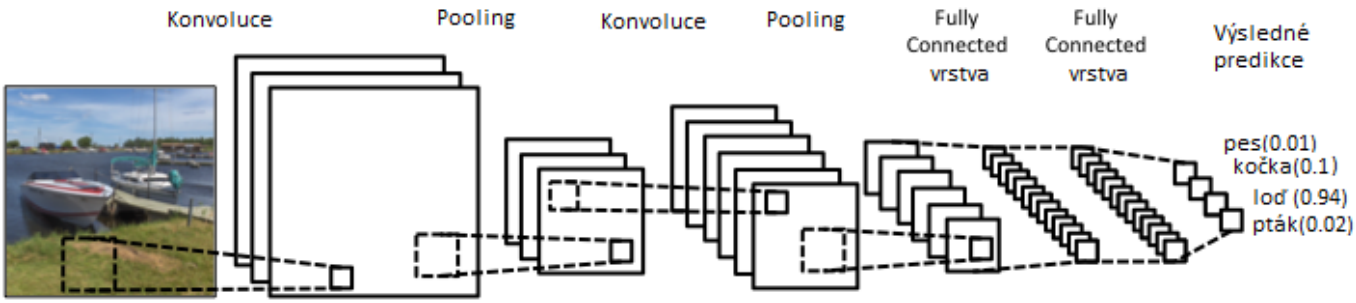
Kvůli povaze podvzorkování (ztráta informace) se vedou diskuze o vhodnosti použití této vrstvy. Možným řešením je nahrazení všech podvzorkovacích vrstev konvolučními vrstvami s velkým krokem. Tento postup ovšem velmi zpomaluje učící fázi a mnohonásobně zvyšuje celkovou komplexnost sítě. Je ovšem možné, že v budoucnu budou podvzorkovací vrstvy vyřazeny z architektur konvolučních neuronových sítí (4).

2.3.4 Detekce konvoluční neuronové sítě

Celý průběh zpracování učící fáze v konvoluční neuronové síti je kombinací výše zmíněných částí a vypadá následovně:

1. Nastavení vah na náhodné hodnoty
2. Prvotní zpracování obrazu - úprava velikosti, popřípadě barvy
3. Zpracování obrazu konvoluční vrstvou
4. Zpracování profiltrovaných hodnot plně propojenou sítí
5. Porovnání vlastního výstupu s očekávaným
6. Zpětná propagace odchylky a výpočet gradientu vah
7. Nastavení nových velikostí vah
8. Opakování celého procesu od kroku 2

Po učící fázi následuje fáze testovací, která uplatňuje pouze kroky 2-5. Celý proces testovací fáze včetně klasifikace a úpravy výstupu pomocí softmaxu je představen na následujícím obrázku:



Obrázek 12 - Průběh klasifikace konvoluční sítě (3)

2.4 Hyperparametry neuronové sítě

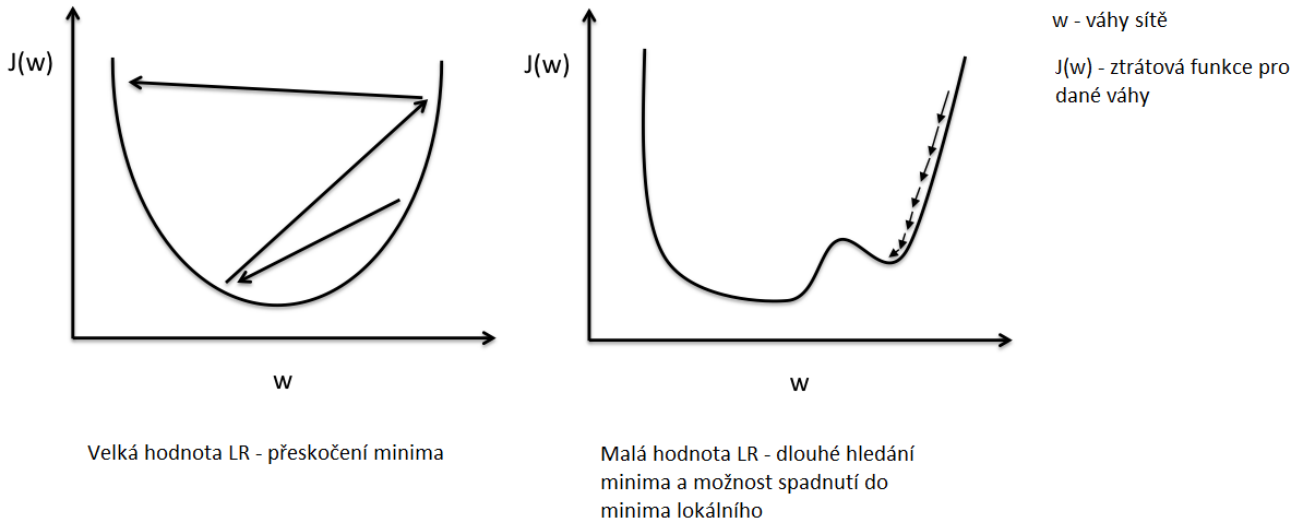
Přestože se CNN může skládat z tisíců parametrů, které jsou vyjádřeny jednotlivými váhami neuronů, existuje několik obecných parametrů ovlivňující celkové chování sítě a její schopnost učit se, klasifikovat, její architekturu a způsob zpracovávání vstupů - takzvané hyperparametry sítě. Tyto parametry bývají upravovány ručně, popřípadě nějakým prohledávacím algoritmem a v principu slouží k optimalizaci celé sítě. Jejich množství není přesně definované - záleží na volbě architektury sítě a navíc mnoho parametrů není esenciálních pro samotnou funkcionalitu sítě, a proto do této chvíle až na výjimky nebyly zmíněny. Správná volba těchto parametrů je ovšem stěžejní pro správnou funkcionalitu sítě a jejich vhodná volba je náročný úkol.

Pravděpodobně nejvlivnější a nejpotřebnějšími hyperparametry jsou ty, které upravují průběh zpětné propagace.

2.4.1 Rychlost učení

Tento parametr (anglicky learning rate) byl již zmíněn a jeho použití je přímočaré: parametr η ovlivňuje rychlost nastavování vah při zpětné propagaci výsledné chyby. Volba této jednotky má velký vliv na učení sítě, přičemž závisí na učící množině. Pokud je tento parametr nastaven příliš velký, sít může "přeskočit" globální minimum a začít kolem něj oscilovat. Naopak při volbě příliš malé hodnoty bude aktualizace vah velmi pomalá a síť bude toto minimum hledat příliš dlouhou dobu přes velké množství epoch (v řádech statisíců/milionů). Rychlost učení je závislá i na velikosti zpětně propagované chyby, která nebývá rovnoměrně rozložená a gradient nabývá velmi různorodých hodnot. Proto je konstantní volba rychlosti učení nepraktická a bývá upravována během učícího procesu i několikrát na základě analýzy poklesu ztrátové funkce.

$$\Delta w(t+1) = w - \eta \frac{\partial E}{\partial w} + \alpha \Delta w(t) \tag{13}$$



Obrázek 13 - Vliv rychlosti učení (LR) na hledání minima (15)

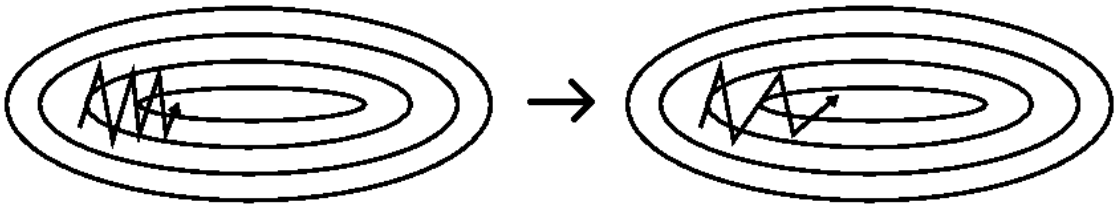
Hodnota rychlosti učení bývá ve většině případů menší než 1. Rozpětí může být od 1e-1 do 1e-6.

2.4.2 Hybnost učení

Hybnost učení (anglicky momentum) je parametr, který se dá snadno zakomponovat do zpětné propagace. Cílem je při nastavování nové váhy vzít v potaz předešlou úpravu váhy a část její minulé změny přidat k aktuálnímu kroku. Vyjádření dané úpravy v rámci rovnice zpětné propagace (11) vypadá následovně:

$$W_{ij}^{vrstva}(t+1) = W_{ij}^{vrstva}(t) - \eta \frac{\partial E}{\partial W_{ij}^{vrstva}} + \alpha \Delta W_{ij}^{vrstva}(t) \qquad (14)$$

kde $\alpha \Delta W_{ij}^{vrstva}(t)$ představuje změnu váhy v daném neuronu a vrstvě v předchozím kroku. Parametr α pak vyjadřuje velikost hybnosti učení. Smyslem této úpravy je zlepšené chování v případě nalezení lokálního minima, respektive snahy dostat se z jeho vlivu. Intuitivně si lze představit tuto hodnotu, tak jak plyne z názvu - jako hybnost nějakého bodu, který prochází krajinu plnou "kopců a údolí", která představují lokální minima, při hledání nejhlubšího údolí, tedy globálního minima. Bez hybnosti by bod "spadl" do první jamky, kde by již zůstal, protože změna parametrů v daném okolí hodnot by vždy vedla k větší odchylce. Při použití hybnosti, tzn. po přičtení části předchozí změny, může být hybnost dostatečná pro opuštění jamky lokálního minima. Hybnost dále zjemňuje oscilace, způsobené prudkými změnami směru gradientu, pokud se blížíme optima po šikmém "údolí".



Obrázek 14 - Ukázka vlivu hybnosti na oscilaci vah (15)

Parametr hybnosti bývá nastaven v rozmezí 0.5 - 0.9.

2.4.3 Rozpad vah

Třetí parametr, který slouží jako přímé rozšíření zpětné propagace je rozpad vah (anglicky weight decay). Smyslem tohoto parametru je penalizace vysokých hodnot vah, které při delším učícím procesu mohou poukazovat na přeučení sítě. Tato penalizace je rozšířením rovnice zpětné propagace a je dána vztahem:

$$W_{ij}^{v}(t+1) = W_{ij}^{v}(t) - \eta \frac{\partial E}{\partial W_{ij}^{v}} - \eta\lambda W_{ij}^{v}(t) \tag{15}$$

Kde λ je multiplikátor rozpadu vah, který s iteracemi klesá. Kombinací s hyperparametrem hybnosti pak dostáváme celkovou rozšířenou rovnici zpětné propagace:

$$W_{ij}^{v}(t+1) = W_{ij}^{v}(t) - \eta \frac{\partial E}{\partial W_{ij}^{v}} + \alpha \Delta W_{ij}^{v}(t) - \eta\lambda W_{ij}^{v}(t) \tag{16}$$

Jak bylo zmíněno výše, tyto parametry nejsou pro funkci esenciální, ale pouze zrychlují a optimalizují učící proces. Zpětnou propagaci je možno rozšířit vlastními heuristickými algoritmy a rovnicemi, jak uzná uživatel za vhodné pro svůj úkol.

Další hyperparametry se již nebudou týkat úprav rovnice zpětné propagace, ale jiných částí neuronové sítě.

2.4.4 Dropout

Parametr dropout se týká plně propojených vrstev. Pokud je do sítě přidán parametr dropout, znamená to, že při učící iteraci jsou některé uzly náhodně vynechány. Přestože se na první pohled může zdát, že je tento krok nelogický, ve skutečnosti slouží k rovnoměrnějšímu rozložení důležitosti vah v celé síti a snižuje hrozbu přeučení sítě. Pokud jsou vynechány klíčové uzly, síť se bude snažit rozprostřít vhodné nastavení vah přes momentálně aktivní uzly, čímž dochází k rozložení citlivých neuronů v rámci celé sítě. Je nutné podotknout, že při každé iteraci jsou vypnuty jiné uzly, jinak by tento parametr ztrácel smysl. Velikost parametru je v rozmezí (0,1). a udává, s jakou pravděpodobností bude každý uzel v rámci iterace deaktivovaný. Pokud tedy nastavíme hodnotu na 0.5, bude v rámci učícího procesu vždy aktivní zhruba polovina neuronů. To ovšem

neznamená, že ostatní neurony jsou zbytečné, jsou pouze aktivní v jiné iteraci. Při testování sítě jsou aktivní všechny neurony.

3 Lokalizace objektů v obraze

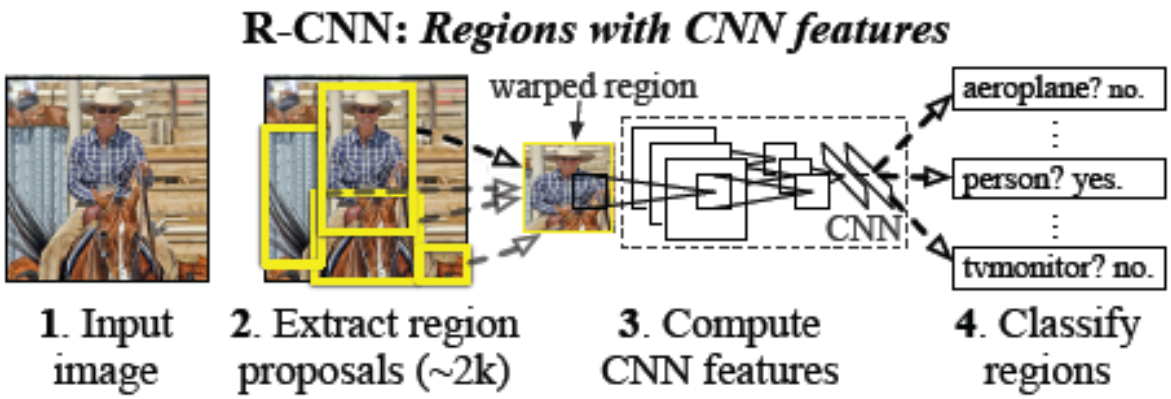
3.1 Stav problematiky lokalizace objektů pomocí neuronových sítí

Neuronové sítě jsou velice efektivní nástroje pro identifikaci objektů a v posledních letech velmi rychle překonávají nejpokročilejší algoritmy založené na specifických analýzách obrazu či jiných metodách (5). Výhodou je bezpochyby jednoduchost jejich návrhu, který se skládá z několika se opakujících segmentů. Univerzálnost, umožňující identifikaci jakýchkoliv tříd objektů, která, na rozdíl od jiných metod, není závislá na předem připravených deskriptorech popisující specifické vlastnosti jednotlivých objektů (kola u aut, oči v obličeji atd.), ale pouze na velikosti a kvalitě učící množiny objektů. V neposlední řadě je výhodou i velmi přesná predikce výsledků.

Problém nastává v lokalizaci samotného objektu v rámci obrazu. Moderní metody, které tento problém řeší, se skládají převážně ze dvou možností - výběr míst v obraze pomocí algoritmů třetích stran na které je uplatněna klasifikace pomocí neuronové sítě pro které se používá zkratka RCNN (region + konvoluční neuronová síť), případně metoda "posuvného okna" kdy vstupem neuronové sítě je neustále se posouvající část obrazu.

3.1.1 RCNN metody

Tyto metody patří k nejúspěšnějším způsobům lokalizace objektů pomocí neuronových sítí a dosahují skvělých výsledků v přesnosti lokalizace i v porovnání s ostatními metodami. Základem je spojení výhod existujících algoritmů a neuronových sítí, čímž je dosažena maximální efektivita. Externí algoritmus nejprve z obrazu extrahuje zajímavá místa, ve kterých by se mohl nacházet objekt. Takovým programem je například často používaný " Selective Search for Object Recognition" (6). Tyto oblasti jsou následně z obrazu vyříznuty a transformovány do potřebné velikosti pro vstup dané konvoluční sítě. Ta ohodnotí každou z těchto oblastí a přiřadí jí skóre podle pravděpodobnosti výskytu objektu - například formou standardní softmax klasifikace. Následně je toto skóre vyhodnoceno podle nastavených kritérií a pozitivní oblasti jsou vyznačeny.

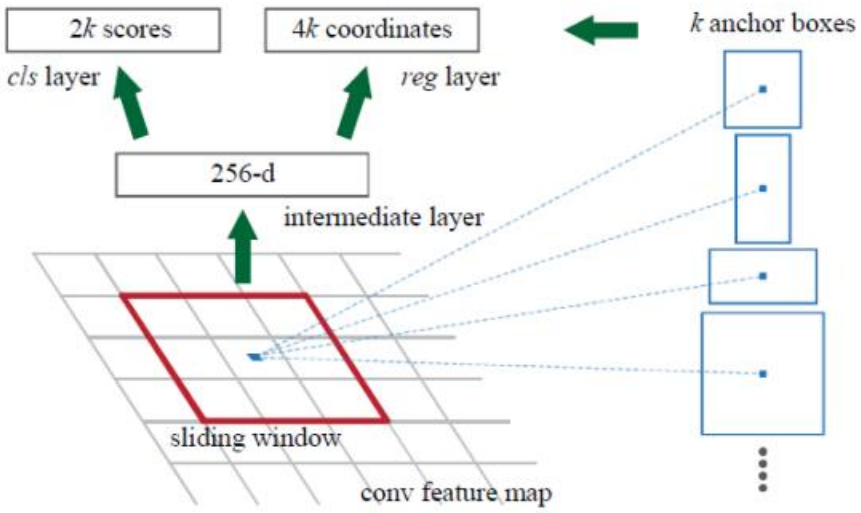


Obrázek 15 - Princip R-CNN metod (12)

Nejrychlejší z těchto metod dosahují rychlosti zhruba 5 snímků za vteřinu, což se blíží zpracování v reálném čase ovšem za cenu použití externích algoritmů pro detekci zajímavých míst v obraze (7).

3.1.2 Metody posuvného okna

Tato metoda je ve svém principu velmi přímočará. Konvoluční síť bere jako vstup určitou část obrazu a tu vyhodnocuje. Zpracováním rozdílných klasifikačních hodnot pro jednotlivé oblasti obrazu vznikne mapa nejpravděpodobnějších míst výskytu daného objektu. Tato metoda silně závisí na volbě posuvného okna, a proto se často používá několik velikostí. To znamená, že pro každou oblast je podstoupeno mnoho výpočtů, přímo závislých od počtu zvolených oken. Pro lokalizaci jediného objektu v obraze je tak třeba zdlouhavého opakujícího se procesu, a proto není tato metoda vhodná pro vyhodnocování polohy v reálném čase, přestože je její přesnost velmi vysoká (8). Nejmodernější a nejrychlejší sítě založené na metodě posuvného okna, jako například síť OverFeat (8), jsou schopny dosahovat rychlosti zhruba 1 snímku za vteřinu.



Obrázek 16 - Princip posuvného okna (7)

3.1.3 Výběr algoritmu pro detekci

Přestože výše zmíněné metody jsou velmi přesné, nelze je kvůli jejich pomalejší rychlosti uplatnit pro lokalizaci objektu z živého videozáběru. Z tohoto důvodu je v této práci připraven prototyp vlastního návrhu využívající samotných vlastností konvolučních sítí. Tento prototyp detekce bude navrhnut v prostředí Caffe (9).

3.2 Caffe framework

Pro návrh sítě byl vybrán software Caffe napsaný v jazyce C++. Tento opensource nástroj obsahuje mnoho standardních knihoven pro prácí a návrh neuronové sítě včetně tvorby konvolučních neuronových sítí. Caffe dále podporuje učení za využití výkonu grafické karty. Tato vlastnost velmi zrychluje testování a analýzu navrhnuté sítě - zrychlení oproti učení pomocí CPU je několikanásobné, podle výkonu grafické karty. Dále díky skutečnosti, že je kód napsán v jazyce C++ a při jeho tvorbě byl kladen důraz na rychlost, je tento framework ideální pro tvorbu sítí, jejichž úkolem je zpracování obrazu v reálném čase. Poslední výhodou je integrace nadstavby v jazyce python, která umožňuje dodatečné úpravy chování sítě, vyhodnocování výstupů nebo i tvorbu vlastních vrstev, které lze snadno implementovat do celkové architektury, podle specifických potřeb uživatele.

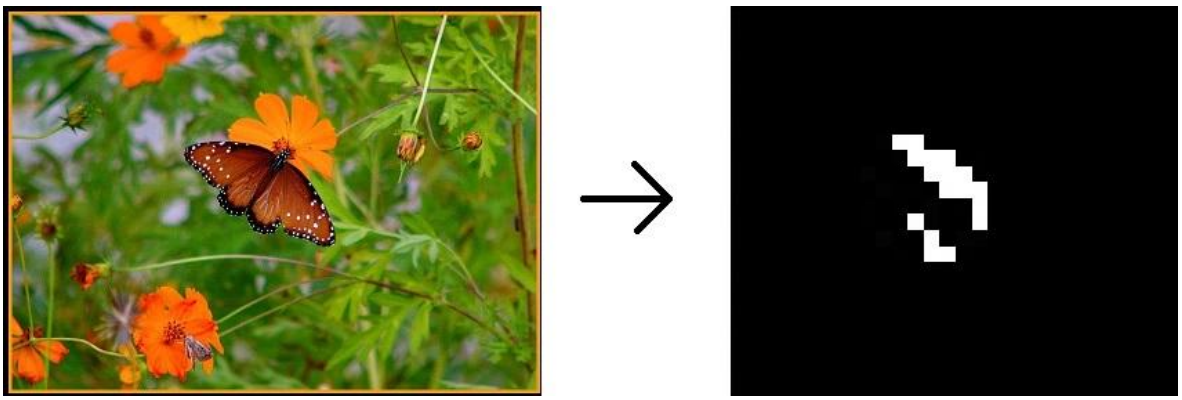
Samotný návrh je založen na vytváření textového souboru (tzv. prototxt) pevně dané struktury pomocí jednoduchých stavebních bloků představující vrstvy neuronové sítě. Informace v tomto souboru zahrnují počet vrstev, jednotlivé parametry vrstev jako počet neuronů, velikost kroku konvoluce, způsob podvzorkování a vstupní data. Druhý textový soubor (tzv. solver) obsahuje informace o hyperparametrech sítě jako je rychlost učení, rozklad vah i doba učení a odkaz na připravený prototext.

Spuštěním caffe aplikace s vhodnými argumenty spustíme učení/testování neuronové sítě na připravené množině dat.

3.3 Vlastní návrh řešení

3.3.1 Motivace

Protože cílem práce je lokalizace objektu (dron) v reálném čase, byla při návrhu uvažována rychlost výpočtu a vykreslení polohy na prvním místě. Základní myšlenkou návrhu je skutečnost, že samotná aktivace neuronů v konvolučních vrstvách může poskytovat informace o poloze objektu:



Obrázek 17 - Příklad informace o poloze objektu uvnitř CNN

Z obrázku vidíme, že síť pomocí naučených vah zvýraznila v jedné ze svých konvolučních map tvar motýla, zatímco odstranila všechny ostatní informace z obrazu. Proto pokud bude tato informace vhodně zpracována plně propojenou vrstvou, bylo by možné síť naučit tuto polohu v rámci vrstvy vyhodnotit jako polohu objektu v obraze. Učení sítě by tak nejen mělo klasifikovat třídu objektu, ale také porovnávat hrubou polohu objektu v obraze s předem známou vyznačenou polohou.

Zároveň je nutné vzít v úvahu strukturu značení polohy objektu. Pro lokalizaci objektů se obecně používají dvě metody. Vyznačení hrubého obdélníku okolo objektu nebo přesné značení pixel po pixelu v obraze viz obrázek - tzv pixelová segmentace. Intuitivně je jasné, že pro rychlý

Obrázek 18 - Používané metody lokalizace objektu v obraze (13)

výpočet je vhodnější vyznačení pomocí obdélníku, protože výstupem polohy budou pouze souřadnice rohových bodů obdélníku, narozdíl od zdlouhavé klasifikace všech pixelů v obraze.

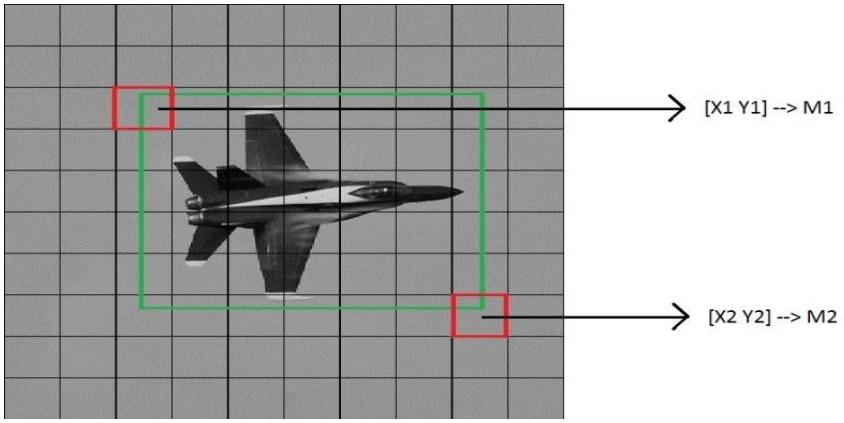
3.3.2 Návrh lokalizace objektu

Základní myšlenkou návrhu je naučit se z učící množiny polohu objektů v obraze. Při návrhu tak bylo třeba vzít v úvahu několik faktorů.

Prvním faktorem je zpracování obrazu. Protože naším cílem je rychlá lokalizace objektu, musí být vstupní velikost obrazu zmenšena, aby síť pracovala s menším množstvím dat. Toto je standardní postup u konvolučních sítí a zvyšuje efektivitu učení. Výstupem lokalizace objektu pak ovšem mají být souřadnice obdélníku. Protože vstupní obraz je transformován na pevně definovanou velikost, musí být i souřadnice v rámci obrazu relativní k samotnému obrazu.

Druhým bodem je počet výstupních tříd, které v tomto případě reprezentují polohu objektu v obraze. Vzhledem k relativnosti polohy k obrazu (viz předchozí bod) se nabízí jako vhodná varianta procentuální poloha rohu obdélníku v osách x a y. Výstup by tak byl tvořen čtyřmi výstupy, kde každý by obsahoval nezávislých 100 tříd (0-100%). To je 10000 možných kombinací na každý roh objektu. Vzhledem k principu neuronové sítě, která vyžaduje dostatečný počet vzorků pro jednotlivé třídy by taková síť potřebovala příliš velkou množinu vstupních dat.

Na základě těchto bodů byl navrhnut následující systém lokalizace. Každý obraz bude tvořen mřížkou, kde každé políčko bude představovat sektor výskytu lokalizačního obdélníku. Poloha těchto sektorů [X_i Y_i] budou kódovací funkcí převedeny na celé číslo M_i, čímž se zredukuje počet výstupů sítě na pouhé dva pro lokalizaci objektu:



Obrázek 19 - Ukázka kódování polohy hran obdélníku

Tímto způsobem dostáváme jednoduchý a rychlý nástroj pro učení polohy objektu v obraze. Výběr velikosti mřížky je posledním krokem a jako vhodný kompromis mezi množství kategorií, kvalitou označení a pozdějším zpracováním informací byl obraz rozdělen do mřížky 10×10. To znamená, že pro každou hranu existuje pouze 100 tříd, ze kterých se bude algoritmus učit odhadnout polohu objektu.

Největší výhodou oproti používaným metodám je především v rychlosti. Zatímco metody posuvného okna vyžadují mnohonásobné výpočty pro každý jednotlivý obrázek a metody typu R-CNN externí algoritmy pro výpočet zajímavých míst, tato metoda potřebuje pro svůj odhad pouze jeden průchod neuronovou sítí pro daný obraz, ovšem za cenu očekávané snížené přesnosti.

3.4 Návrh neuronové sítě

Vlastní neuronová síť bude klasická verze konvoluční sítě. Návrh je rozdělen do několika kroků vzhledem k vyhodnocení výsledku a struktury konvoluční sítě. Prvním krokem je zpracování obrazu před vstupem do konvoluční sítě, druhým krokem je návrh konvoluční sítě a třetím krokem je návrh plně propojené sítě.

3.4.1 Vstupní obraz

Protože barevný obraz obsahuje jasovou složku pro každou základní barvu (RGB obraz), byla by konvoluční část výpočtu na barevném obraze 3× delší, než v případě použití černobílého obrazu. Pro vstup konvoluční sítě je tedy obraz nejprve upraven do odstínů šedi. Tato volba klade menší nároky na paměť, rozličnost učících dat (barva může způsobit přeučení), na množství vah (3× počet pro barevný obraz) i na zpracování plně propojenou vrstvou aniž by přinášela výrazné zlepšení přesnosti.

Druhým bodem úpravy vstupního obrazu je jeho velikost. Byl hledán kompromis mezi dostatečně velkým vstupním obrazem, který neztratí příliš mnoho detailů deformací a rychlostí zpracování. Čím menší obraz, tím méně konvolučních a podvzorkovacích vrstev stačí pro extrakci vlastností objektu a vypočtení jeho polohy - viz teoretická část.

Testováním byla jako nejvhodnější velikost poskytující dostatečný kompromis mezi rychlostí a přesností zvolena hodnota 256×256 pixelů.

3.4.2 Návrh konvoluční vrstvy sítě

Pro návrh sítě bylo zvoleno jednoduché kritérium. Průběžné snižování velikosti pomocí konvoluce/podvzorkování s co nejmenší ztrátou informace. Proto byl zvolen co nejmenší krok posunu i velikost podvzorkování. Krok konvoluce byl volen co nejmenší, ale tak, aby splňoval účel konvoluce pro hledání zajímavých rysů. Nakonec byla velikost jádra nastavena na 3×3 a 4×4. Podvzorkování bylo vždy nastaveno na krok 2×2 - po každém podvzorkování tak docházelo ke snížení rozlišení o polovinu. Typ algoritmu podvzorkování byl výběr maximální hodnoty pro ostré zvýraznění detekovaných příznaků. Tímto postupem došlo k profiltrování vstupu až ke konečné hodnotě vzhledem k dříve odvozeným vzorcům:

1. Vstup 256×256
2. Konvoluce (1): 256×256 -> 254×254
3. Podvzorkování (1): 254×254 -> 127×127
4. Konvoluce (2): 127×127 -> 124×124
5. Podvzorkování (2): 124×124 -> 62×62
6. Konvoluce (3): 62×62 -> 60×60
7. Podvzorkování (3): 60×60 -> 30×30
8. Konvoluce (4): 30×30 -> 28×28
9. Podvzorkování (4): 28×28 -> 14×14

Každá vrstva je tvořena určitým počtem konvolučních jader (viz kapitola o konvolučních sítích), které jsem se rozhodl s každou další vrstvou zdvojnásobit. První vrstva obsahuje 16 filtrů, poslední pak 128.

Protože lokalizace objektu se skládá z vybrání vhodného regionu z mřížky o rozměrech 10×10, intuitivně jsem zvolil jako poslední konvoluční vrstvu s výstupem 14×14, která se této mřížce podobá. Výsledky konvoluční vrstvy jsou následovně zpracovány v plně propojené neuronové síti.

3.4.3 Návrh plně propojené vrstvy

Vzhledem k výstupu předchozí vrstvy skládající se ze 128 různých map aktivovaných podle konvolučních jader o velikosti 14×14 by bylo vhodné použít vrstvu, kde by počet neuronů

přesahoval počet map pro jistou míru adaptace. Proto vstupní vrstva proto byla vybrána o velikosti 200 neuronů.

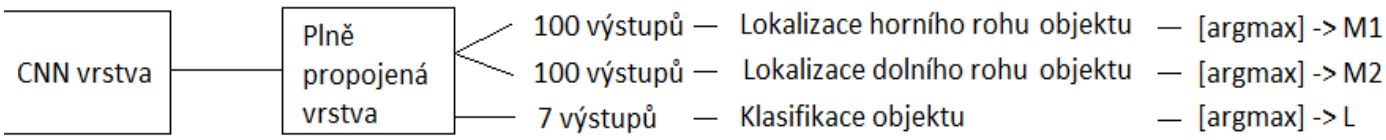
Vzhledem k malé množině učených kategorií a co nejrychlejšímu zpracování informací, bude síť testována s jednou skrytou vrstvou o velikosti v rozmezí 0-1000 neuronů. Jednotlivé výsledky budou porovnány pro vybrání optimální velikosti vrstvy.

Tyto dvě vrstvy budou dále testovány s použitím několika hodnot hyperparametru dropout, který velmi silně ovlivňuje učení a tím i následné chování celé sítě.

Výstupní vrstva plně propojené sítě se skládá ze třech samostatných částí a vychází přímo z navrženého zpracování lokalizace objektu. Samostatností se myslí tři různé skupiny výstupů, ne pouhé množství neuronů. První dvě části jsou lokalizace objektu. Jak bylo zmíněno výše, náš výstup pro lokalizaci byl zjednodušen na pouhé dvě hodnoty - ty přesně tvoří první tyto části výstupu. Nejsou součástí jednoho společného výstupu, protože každá z těchto hodnot musí být zvlášť vyhodnocena. Každý z nich je tvořen 100 třídami, což odpovídá mřížce lokalizace. Hodnoty pro jednotlivé třídy jsou upraveny pomocí funkce softmax do pravděpodobnostního rozdělení pro správně funkční zpětnou propagaci.

Druhou částí je samotné rozpoznání objektu. Toto je standardní výstup konvolučních sítí a byl detailně popsán v teoretické části. Skládá se z počtu výstupů podle klasifikovaných tříd, které jsou aktivované opět pomocí funkce softmax. Přestože se snažíme pouze o lokalizaci dronu, je tato vrstva nutná, abychom odlišili dron od okolí a nesnažili se vykreslit objekt v případě, že na obraze žádný dron není. Navíc se při návrhu ukázalo, že pokud jsme měli pouze dvě výstupní klasifikační třídy, docházelo k silnému přeučení sítě pro lokalizaci kvůli pozadí. Důvodem bylo, že i v případě obrazu bez drona bylo nutné nějak definovat hodnoty pro lokalizaci - ty byly pevně nastaveny na hodnotu 0 pro oba případy. Tyto hodnoty tak převládaly oproti ostatním a silně ovlivnili učení sítě. Řešením tohoto problému bylo nahrazení okolí dalšími několika třídami objektů, které se naučila síť rozpoznávat a lokalizovat. Toto rozšíření považuji za praktické, protože pouze rozšiřuje schopnosti sítě za nulového zvýšení komplexnosti. Finální počet tříd byl určen na 8. Tato hodnota není však pro cíl práce příliš důležitá a pouze určuje, pro kolik typů objektů byla síť naučena a může být kdykoliv změněna.

Při vyhodnocení výsledků je pro každou z těchto tříd vybrána maximální hodnota, která představuje největší "jistotu" sítě, že daný výsledek je správný. Vyhodnocení plně propojené vrstvy je tedy následující:



Obrázek 20 - Grafická představa výstupu neuronové sítě

3.4.4 Hyperparametry sítě

Poslední část návrhu sítě se týká obecných hyperparametrů sítě. Tyto parametry byly nastavovány na základě průběžného testování pro dosažení co nejoptimálnější rychlost učení a přesnosti. Síť obsahovala všechny parametry zmíněné v teoretické části. Pro přehlednost jsou tyto parametry zobrazeny v následující tabulce. Parametry jsou popsány i v angličtině, tak, jak jsou inicializovány v programovém nástroji Caffe pro výpočet neuronové sítě. Výchozí hodnoty hyperparametrů, které se pak budou upravovat na základě analýzy přesnosti jsou následující:

Hyperparametr	**Hodnota**
Rychlost učení (learning rate)	0.0001
Hybnost (momentum)	0.9
Rozklad vah (weight decay)	0.00005
Dropout	0-0.7
Počet vzorků v iteraci (batch size)	64
Počet učících iterací (max iter)	10 000
Celkový počet epoch	156

Tabulka 1 - Hodnoty hyperparametrů

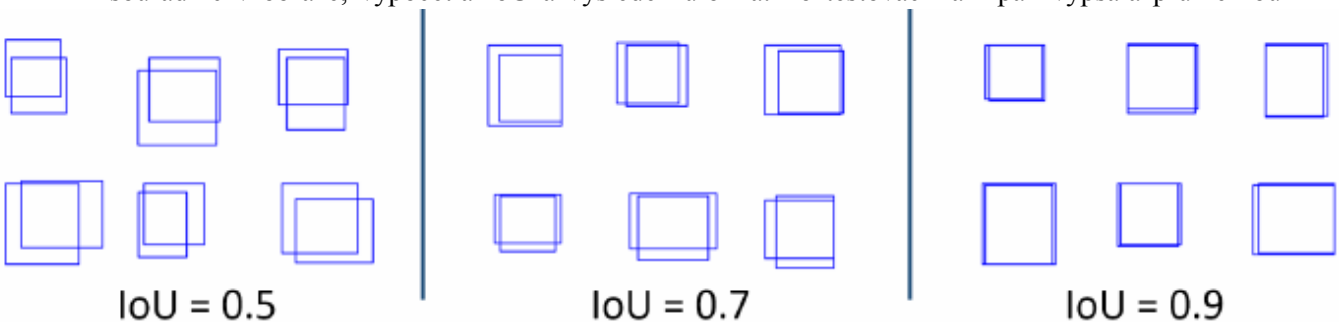
4 Testovací data

4.1.1 Výstup sítě a vyhodnocovací vrstvy

Testování funkcionality a přesnosti sítě stejně jako analýza přesnosti probíhala na množině obrázků, které nepatřily do učící množiny a obsahovaly snímky dronů. Samotné finální zpracování sítě se ovšem týká vyhodnocení obrazu a vykreslení polohy dronu v rámci real-time videa.

Přestože Caffe framework kromě výpočtu umožňuje porovnání výsledků se skutečnou hodnotou v případě klasifikace, nepodporuje porovnání lokalizace objektu. Pro porovnání vyznačené oblasti je nejvhodnější způsob pomocí překrývané plochy mezi skutečností a odhadem. Této metodě se anglicky říká "Intersection over Union" (dále IoU) a standardně se používá pro vyhodnocení přesnosti lokalizačních algoritmů. Hodnota IoU vyjadřuje podíl mezi překrývanou plochou a sjednocenou plochou odhadované lokace a skutečné polohy objektu.

Pro toto vyjádření bylo nutné vytvořit vlastní vyhodnocovací vrstvu v jazyce Python. Vstupem této vrstvy byly vypočtené hodnoty polohy obdélníku neuronové sítě a hodnoty obdélníku skutečné polohy. Vyhodnocovací vrstva nejdříve hodnoty z neuronové sítě dekódovala do souřadnic v obraze, vypočetla IoU a výsledek uložila. Po testovací fázi pak vypsala průměrnou



Obrázek 21 - Příklad analýzy přesnosti pomocí IoU metody

hodnotu IoU pro celý testovaný balíček obrazů. Tímto způsobem bylo možné snadno analyzovat funkcionalitu celé sítě. V praxi se jako hodnota, která dostatečně odpovídá přesné lokalizaci používá práh od 0.5 do 0.7 (může být vyšší pro striktnější kontrolu, ale vzhledem k přesnosti naší sítě byl zvolen práh 0.5).

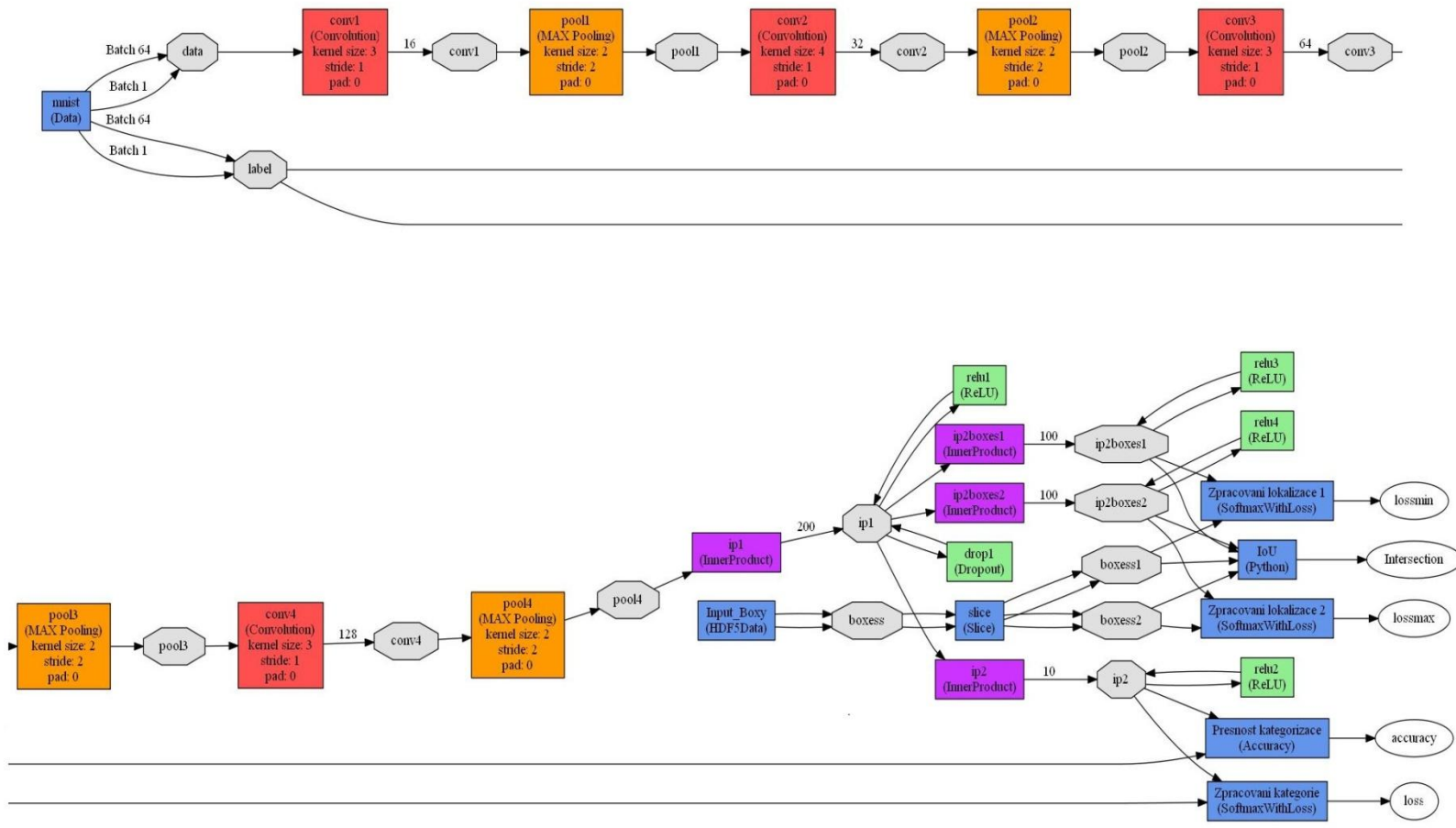
Tato vrstva není připravena pro zpětnou propagaci a používala se pouze v testovací fázi pro číselné zpracování výsledků.

Další vrstvy, které slouží pro vyhodnocení výstupu jsou ztrátové funkce (loss functions), které jsou použity pro nastavení vah algoritmem zpětné propagace a jsou standardní součásti knihovny Caffe. Tyto funkce jsou přímo napojeny na výstupy plně propojené vrstvy a porovnávají vypočtenou hodnotu kategorie i polohy obdélníku se skutečnou. Z těchto informací tato vrstva vypočítá požadovanou odchylku pro zpětnou propagaci viz kapitola 2.2.3 .

Poslední částí sloužící pro získání informace o přesnosti kategorizace je vrstva zpracovávající přesnost detekce(v angličtině accuracy layer). Ta je napojena stejně jako vrstvy pro ztrátové funkce a jejím výstupem je jednoduše poměr správně označených objektů k nesprávně označeným. Výstup této funkce slouží pro analýzu funkcionality i zpracování výsledků, přestože správné rozpoznání objektu není pro tuto práci hlavní úkol.

4.1.2 Shrnutí architektury sítě

Celá síť se skládá z konvoluční vrstvy, plně propojené vrstvy a výstupních vrstev sloužící pro vyhodnocení výsledků a případně zpětné propagaci těchto hodnot. Pro přehlednost je celá architektura sítě vykreslena na následujícím obrázku:



Obrázek 22 - Celkový architektura sítě

4.2 Příprava učících dat

Caffe framework vyžaduje speciální formát vstupních dat pro co nejrychlejší zpracování obrazu - formát .lmdb. Dále obsahuje podporu pro vytvoření lmdb souboru spolu s kategorizací zpracovaných obrazů. Pro tvorbu tohoto šifrovaného souboru je nutné mít připravený textový soubor obsahující relativní cestu k jednotlivým obrázkům a číslo definující její kategorii (v našem

případě tedy čísla od 0 do 7). Pomocí několika příkazů Caffe aplikace je pak vytvořen soubor .lmdb, který obsahuje všechny obrázky převedené do vybrané velikosti a barvy - v našem případě do odstínu šedi. Zadané kategorie jsou zpracovány a zvlášť uloženy pro porovnání s výstupem sítě.

Pro učení neuronové sítě bylo potřeba vytvořit množinu obrázků, na kterých je lokalizovaný objekt a neuronová síť pak porovná svůj výstup se správným výstupem a podle rozdílu metodou zpětné propagace upraví vnitřní váhy mezi neurony. Vzhledem k tomu, že neuronové sítě potřebují pro správné naučení detekce velké množství obrázků (v řádech tisíců - čím více, tím lépe, neuronová síť pak lépe generalizuje potřebné vlastnosti objektů), je velmi náročné vytvářet takovou množinu manuálně. Proto bylo využito databáze IMAGENET, která obsahuje miliony obrázků rozdělených do větvících se kategorií.

Přestože tato veřejně dostupná databáze umožňuje po registraci stažení jakékoliv skupiny obrázků, v době psaní této práce měla databáze problémy s účty a neumožňovala volné stažení. Stránka byla ovšem schopna generovat textový soubor obsahující URL všech obrázků z vybraných kategorií. Připravil jsem proto skript, který byl s pomocí tohoto souboru schopný stáhnout daný set obrázků a rovnou označkovat jako zvolenou třídu. Bohužel databáze Imagenet neobsahovala kategorii dron, a proto byla využita pouze pro ostatní (vedlejší) kategorie, které především sloužili pro analýzu funkcionality a pro demonstraci adaptability sítě na různé objekty.

Pro kategorii dron jsem měl k dispozici několik videí letícího drona od katedry kybernetiky, které jsem s pomocí vytvořeného skriptu v jazyce python rozdělil na velké množství obrazů. Vzhledem k velké podobnosti získaných obrázků bylo nutné tyto data profiltrovat a snažit se vybrat vhodné vzorky. Ty byly následně pomocí matlab skriptu zrcadleny podle svislé osy, aby se zvýšila učící množina. Pro kategorii dron tak bylo vytvořeno cca 2000 obrázků.

Druhá část přípravy dat se týkala samotné lokalizace. Každý vzorek, ze kterého se síť učila, musel mít definovanou polohu hledaného objektu. Pomocí skriptu, vytvořeného v programu Matlab, byly hledané objekty manuálně označeny a poloha rámečku zašifrovaná do dvou hodnot (levý horní a pravý dolní roh) podle navržené funkce.

Tyto hodnoty byly uloženy a převedeny do formátu HDF5, který je podporovaný Caffe aplikací. Důvodem uložení do jiného formátu je ten, že Caffe nepodporuje učení polohy,umí ovšem třídit data vložená pomocí HDF5 formátu a pracovat s nimi pomocí speciální vrstvy (anglicky slice layer) uvnitř prototext souboru. Tato speciální vrstva umožňuje transformovat vektory, v tomto případě bylo potřeba transformace z [M1 M2] -> [M1], [M2] . Tímto postupem bylo umožněno pracovat s lokalizačními daty a porovnávat je s výstupem stejným způsobem jako v případě tříd objektů.

Všechny vypracované a výše zmíněné skripty jsou pro srozumitelnost práce součástí přílohy.

4.3 Zpracování živého přenosu a vykreslení lokalizace

Zpracování videa, jeho vyhodnocení a následné vykreslení bylo opět řešeno pomocí nadstavby Caffe, která umožňuje rozšiřovat možnosti práce s neuronovými sítěmi pomocí jazyku Python. Tato nadstavba obsahuje metody pro inicializaci sítí včetně nastavení připravených vah naučených podle našich potřeb a architektury sítě.

Algoritmus využívá další python knihovny, konkrétně cv2 a numpy pro práci se vstupními daty a pro vykreslení obrazu spolu s označenými objekty.

Nejprve je inicializovaná síť s naučenými váhami. Následně je vytvořeno spojení s vybranou připojenou kamerou a načten první snímek - tímto krokem se spouští cyklus načítání obrazu. Obraz kamery je poté transformován na vstupní velikost sítě, převeden do černobílé barvy a načten sítí jako její vstup. Poté dochází k výpočtu uvnitř naučené neuronové sítě. Všechny výstupy jsou extrahovány a z vektorů jsou vybrány maximální hodnoty. Ty určují nejpravděpodobnější typ objektu a jeho odhadnutou polohu. V případě polohy je nutné dešifrovat výstup kódovaný výstupními neurony a převést ho na hodnoty odpovídající souřadnicím v obraze. Vzhledem k transformaci obrazu je nutné tyto hodnoty zpětně upravit do původních velikostí pro nezkreslené zobrazení. Posledním krokem je vykreslení obdélníku do obrazu ve vhodném sektoru a následné zpracování dalšího snímku. Celý cyklus se opakuje, dokud ho uživatel neukončí klávesou ESC.

Celý průběh algoritmu je pro přehlednost znázorněn v jednoduchém pseudokódu na následujícím výpisu:

```
main function:
init net(„Path/to/learned/weights")
init camera
while camera.videoStreamOn() == True:
    width,height = camera.getResolution()
    image = camera.getImage()
    key = keyboard.waitForKey()
    image.showImage()
    neuralNet.input = image.resize(256,256,1)
    output = neuralNet.computeOutput()
    boxTopCorner = output.getBoxOutput[1].argmax()
    xMin,yMin = boxTopCorner.unhash()
    boxDownCorner = output.getBoxOutput[2].argmax()
    xMax,yMax = boxDownCorner.unhash()
    image.drawRectangle(xMin,yMin, xMax,yMax,weight, height)
    if key = keyboard.ESCAPE:
        break
    end if
end while
camera.releaseVideo()
end function
```

Pseudokód lokalizace v reálném čase

4.4 Testování a výsledky sítě

Pro samotné testování přesnosti sítě byla vytvořena testovací skupina obrázků, které nepatřili do množiny, ze které se síť učila. Obrázky pocházeli ze všech naučených kategorií. Vzhledem k námětu práce byl kladen hlavní důraz na objekty typu dron a proto obrázky této kategorie tvořili polovinu testovaných vzorků. Každý obrázek měl definovanou polohu stejně jako v případě učící množiny pro zjištění přesnosti pomocí metody IoU skrze vlastní definovanou vrstvu.

Testování probíhalo pro několik variant neuronové sítě. Varianta se skrytou vrstvou o různé velikosti a varianta bez skryté vrstvy. Pro každou variantu byl nastaven různý dropout a dílčí výsledky byly porovnány.

Testování sítě bez skryté vnitřní vrstvy v plně propojené vrstvě sítě je uveden v tabulce 1. Obsahuje různé velikosti dropout parametru a průměrné IoU všech přes všechny testované obrázky. Rychlost výpočtu byla pro tyto varianty konstantní, protože parametr dropout pouze ovlivňuje způsob, jakým se síť učí rozpoznávat objekty a neovlivňuje rychlost výpočtu sítě (viz kapitola dropout).

Dropout	0	0.3	0.5	0.7
IoU průměr	0.440278	0.521013	0.522777	0.472392
IoU>0.5	42/100	51/100	52/100	47/100

Tabulka 2 - Vliv parametru Dropout

Z tabulky 1 je vidět, že vhodné nastavení dropout parametru vedlo k nárůstu průměrné hodnoty IoU pro testované obrázky nad úroveň 0.5, což lze považovat za dostatečnou přesnost pro úspěšnou lokalizaci objektu.

Pro zajímavost byla do sítě přidána jedna konvoluční vrstva navíc, aby bylo vyzkoušeno, zda nedojde k nárůstu přesnosti rozpoznávání. Místo 4 vrstev tak síť obsahovala 5 konvolučních a podvzorkovacích složek a výstup této vrstvy je obraz o rozměrech 6×6. Vrstva byla testována s identickými hyperparametry jako nejlepší varianta z předchozích testů (tedy dropout 0.5,0.3) . Porovnání výsledků je následující:

Počet konvolučních vrstev	4	5
IoU (dropout 0.3)	0.521013	0.491527
IoU (dropout 0.5)	0.522777	0.502554
Průměrná rychlost (ms/obrázek) CPU	61	78

Tabulka 3 - Porovnání přesnosti s rozdílným množstvím konvolučních vrstev

Z výsledku je patrné, že zvýšení konvolučních vrstev v našem případě vedlo ke snížení přesnosti lokalizace i zpomalení zpracování obrazu. Další testování tedy opět probíhalo pro sítě, jejichž výstup z konvoluční vrstvy více odpovídal počtu lokalizačních sektorů v obraze.

Následně byly analyzovány různé velikosti skryté složky plně propojené vrstvy, která byla rozšířena o jednu skrytou vrstvu (v předchozích případech byla plně propojená vrstva tvořena pouze vstupními a výstupními neurony, které byly plně propojeny). Velikost hodnoty dropout byl globálně nastaven na 0.5, protože vykazoval největší zlepšení přesnosti sítě.

Velikost skryté vrtsvy - počet neuronů	200	500	700
IoU průměr	0.491398	0.485737	0.498860
IoU>0.5	48/100	46/100	44/100

Tabulka 4 - Lokalizace různých variant plně propojené vrstvy

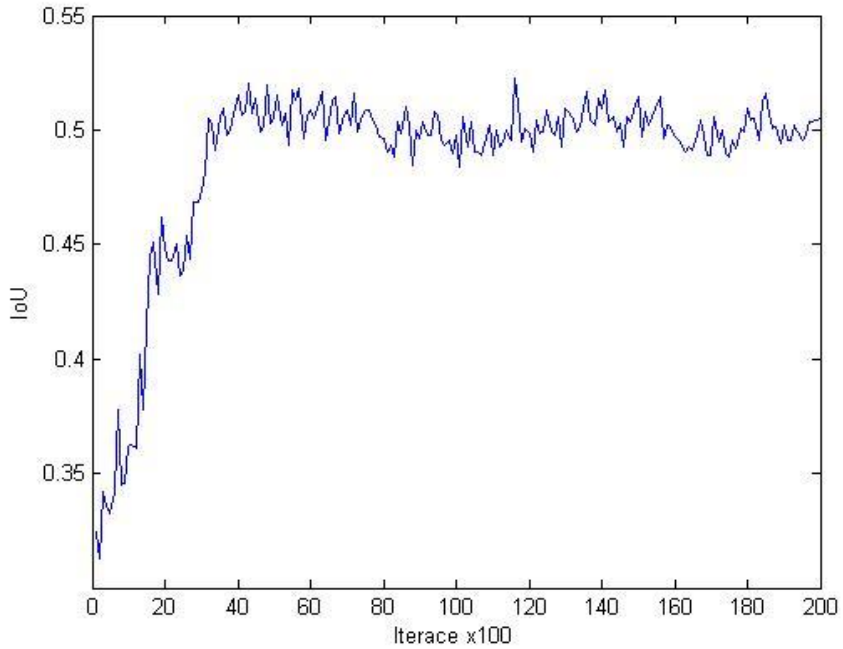
Nejlepších výsledků tak dosahovala překvapivě síť bez rozšířené skryté vrstvy. Větší, plně propojená vrstva, navíc snižovala rychlost výpočtu o zhruba 10ms. Přestože se může jevit takové zpomalení jako nepodstatné, v případě zpracování živého přenosu může 10ms znamenat snížení rychlosti zpracování i o několik snímků za vteřinu. Výsledky přesnosti všech verzí jsou ovšem

velmi podobné a finální úpravy sítě neměli na její funkcionalitu markantní vliv. Rychlost tedy byla v těchto případech rozhodující parametr.

Pro jistotu bylo ale množství učících iterací pro několik testů zvýšeno na dvojnásobek (tedy 20 000 iterací - cca 300 epoch), zda nedojde k výraznějšímu zlepšení přesnosti sítě. Testovány byly nejlepší varianty z předchozích výsledků se skrytou vrstvou, které by zároveň mohli vykazovat největší změny díky velikosti plně propojené vrstvy, která vyžaduje vyšší dobu učení:

Velikost skryté vrtsvy - počet neuronů	200	500
IoU průměr 10000 iterací	0.505069	0.499264
IoU průměr 20000 iterací	0.508743	0.507948

Tabulka 5 - Vliv doby učení na přesnost sítě



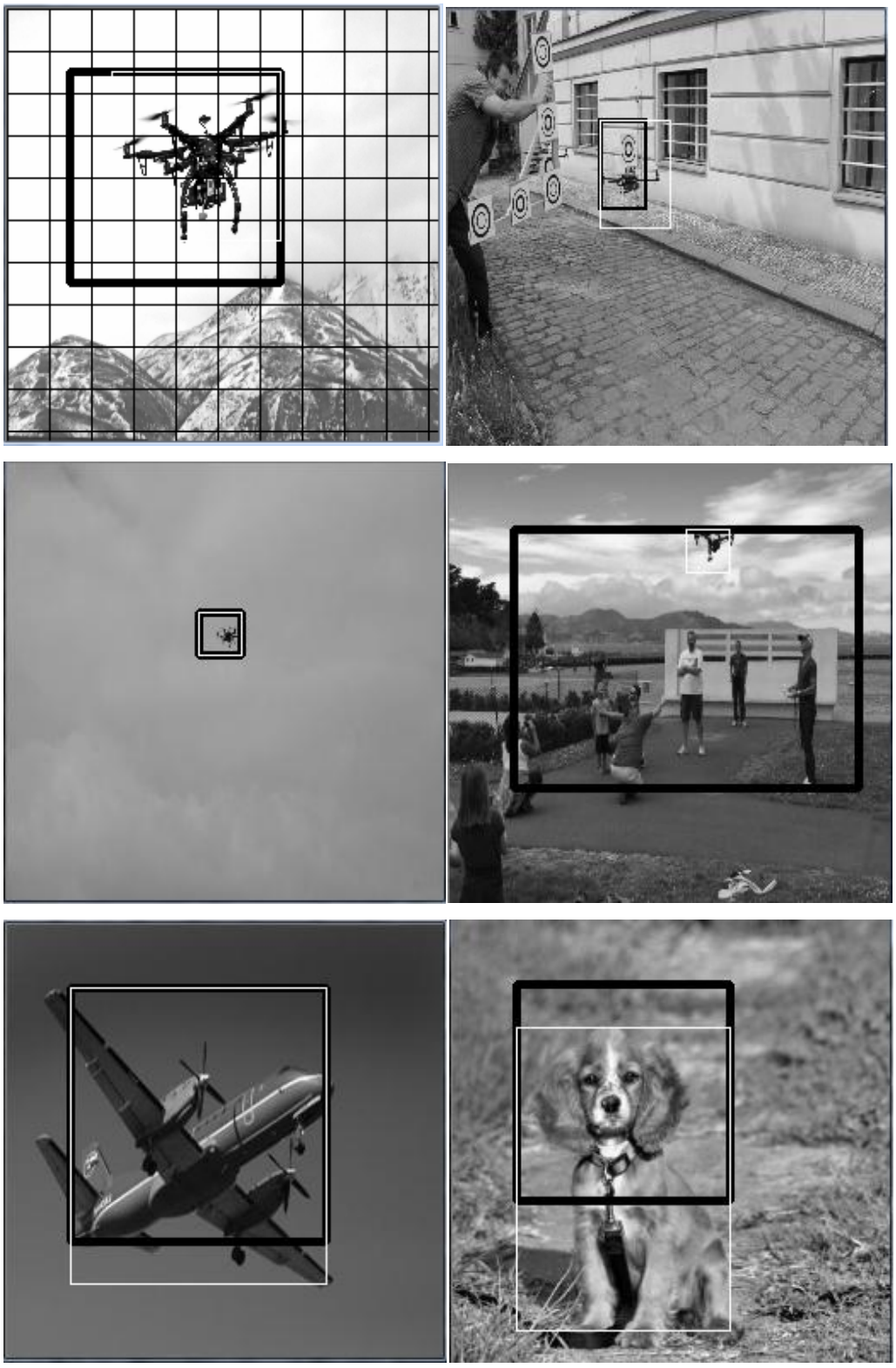
Obrázek 23 - Ukázka relativní stability výsledků po prvních 10000 iterací

Z tabulky a grafu hodnoty IoU pro skrytou vrstvu o velikosti 500 neuronů je patrné, že zvýšení učící doby nevede ke zvýšení přesnosti sítě, hodnota spíše fluktuuje kolem známých hodnot. Podobné výsledky byly vidět i u ostatních verzí sítě s menší skrytou vrstvou Jako nejlepší varianta se nakonec ukázala síť bez rozšířené skryté vrstvy s parametrem dropout nastaveným na hodnou 0.5.

Rychlost všech variant je podle předpokladu vysoká a i při testování pouze za použití výpočetního výkonu CPU (bez využiti zrychlení výpočtem GPU) dosahovala rychlosti zhruba 60-80ms/obrázek. Tato rychlost je pro vykreslení lokalizace v reálném čase dostačující, přesto je asi

nejvhodnější pracovat se sítí, která má z vybraných variant minimální dobou zpracování - tedy varianta bez rozšířené skryté vrstvy s rychlostí 61ms/obrázek.

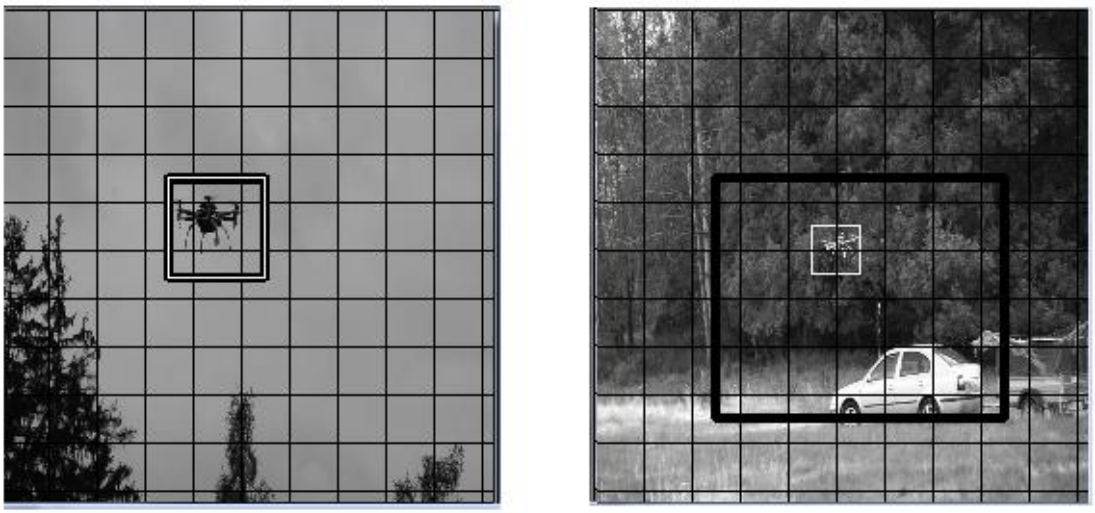
4.5 Ukázky lokalizace objektů



4.6 Shrnutí výsledků

Přestože výsledky ukazují, že je síť schopna najít a úspěšně lokalizovat objekt v obraze, téměř polovina objektů z testované množiny neprošla prahem IoU>0.5, což byl práh minimální úrovně úspěšné lokalizace.

Při analýze jednotlivých obrázků bylo zjištěno, že síť měla především problémy s velmi členitým obrazem obsahující velké množství detailů (například lokalizace drona v lese) a objektů, které se ve své kategorii příliš nepodobali učící množině:



Obrázek 24 - Ukázka správné a velmi špatné lokalizace dle okolí

Tyto problémy jsou velmi pravděpodobně způsobeny menší velikostí učící množiny, ze které se síť není schopna dostatečně kvalitně naučit definující rysy jednotlivých objektů. Pro budoucí účely by tedy bylo vhodné znatelně rozšířit množství připravených vzorků.

Rychlost výpočtu byla dostačující a průměrně dosahovala rychlosti mezi 16-17 snímky za vteřinu a to při použití CPU k výpočtu lokalizace. Taková rychlost je vhodná pro zpracování videa v reálném čase.

Rychlost výpočtu byla následně ověřena použitím vypracovaného skriptu, který byl schopný vykreslovat polohu objektu do video-výstupu webkamery. Tyto výsledky není možné smysluplně představit v tištěné verzi, a proto je v příloze skript pro demonstraci lokalizace drona z videa, tak skript umožňující vykreslení polohy do obrazu webkamery.

4.7 Návrhy na zlepšení a návaznost na budoucí práce

Hlavním cílem této práce bylo otestovat principy neuronových sítí pro lokalizaci objektů v reálném čase a je mnoho částí, která by byla v budoucnu vhodná pro vylepšení. Ty se mohou týkat

zpracování dat, lepší způsob lokalizace nebo větší přesnosti lokalizace. Pro inspiraci v případných budoucích navazujících prací budou jsou možná vylepšení rozebrány v následujících odstavcích.

4.7.1 Augmentace a sběr dat

Hlavním nedostatkem této práce je malá množina učících obrazů, která je silně závislá na ruční přípravě vyznačených oblastí, kde se objekt nachází. Tento nedostatek velmi ovlivňoval schopnosti sítě naučit se obecné znaky objektů a snižoval tak přesnost jak v kategorizaci, tak v lokalizaci.

Řešením tohoto problému by mohlo být využití účinných, i když pomalých lokalizačních algoritmů (například jiné verze neuronových sítí) s vysokou přesností pro automatické vyznačení objektů. Informace z těchto programů by byly následně zpracovány a naučeny vlastní sítí. Přestože by jistě šlo o pomalou metodu, byla by praktičtější než ruční tvorba učící množiny. Problémem této metody je chybovost algoritmů, které by v rámci tisíců klasifikací zanášely jistou chybu do připravených dat. Tato chyba je i přes velkou přesnost stále vyšší než chyba lidská, a proto je nutné ověřit výsledky těchto metod ruční korekcí.

Dalším řešením je využití vytvořených knihoven obrazů včetně těch, které nabízejí informaci o poloze objektu, jako poskytuje stránka Imagenet pro mnoho svých kategorií. Nevýhodou je naprostá závislost na informacích, které třetí strana poskytuje a omezené množství kategorií.

Vhodnější způsob přípravy dat by měl výrazný vliv na zlepšení přesnosti sítě za cenu minimální úpravy architektury sítě. Změnou učících dat je také možné naučit síť pracovat s jinými objekty, než byly použity v této práci. Vzhledem ke skutečnosti, že síť byla připravena na menší množství kategorií, by měl počet nově zvolených kategorií objektů odpovídat počtu kategorií objektů v této práci (cca 10 kategorií). Pro vyšší počet by bylo vhodné zvětšit velikost plně propojené vrstvy, popřípadě přidat více filtrů pro jednotlivé vrstvy konvoluční části. Tato modularita neuronových sítí je mimo jiné jejich největší výhodou a poukazuje na jejich širokou uplatnitelnost v budoucnosti při řešení složitých problémů rozpoznávání a lokalizace.

4.7.2 Detekce více objektů v obraze

Detekce a vyznačení více objektů v obraze lze v této práci jednoduše realizovat vybráním dalších nejvyšších hodnot z výstupních vektorů neuronové sítě, které se vzájemně nepřekrývají a zároveň překračují určitou hodnotu jistoty. Toto řešení není příliš robustní a mohlo by docházet k nesmyslným vyhodnocením polohy.

Navazující práce by se proto mohla zabývat rozšířením práce o kvalitní rozpoznání více objektů v obraze. Způsobem řešení by mohlo být rozdělení obrazu do několika podsektorů, pro které by byly nezávisle vypočteny polohy objektů - tím by se počet lokalizovaných objektů mohl

rozšířit o množství těchto sektorů. Tato úprava by však vyžadovala velké zásahy do architektury sítě i způsobu přípravy učících dat, velmi by však zvýšila využitelnost v praxi.

4.7.3 Nezávislost na Caffe framework

Caffe je univerzální framework pro návrh a pro testování funkcionality neuronových sítí. Jeho instalace a příprava je ovšem poněkud složitá a neintuitivní a pro samotné používání existuje pouze velmi základní úroveň dokumentace. To limituje využití sítí, které jsou v tomto prostředí tvořeny. Export navrhnuté sítě pro samostatné spouštění, nezávislé na prostředí Caffe, by tak bylo vítané. Největší úskalí by v tomto případě bylo napsání rychlého kódu, schopného využívat modul CUDA či OpenCL pro výpočty zpětné propagace na GPU.

5 Závěr

Cílem práce bylo prostudovat metody pro lokalizaci dronů pomocí neuronových sítí a poté navrhnout síť, která by byl schopna tyto objekty detekovat a lokalizovat v kamerovém obraze. S tím souvisel i návrh vlastního algoritmu pro lokalizaci, příprava dat pro vyhodnocení výsledků a následná analýza dosažených výsledků.

Pro tento účel byla navrhnuta architektura vlastní neuronové sítě, včetně algoritmu schopného lokalizovat objekt, dron, v reálném čase. Využití neuronové sítě zjednodušil komplikovanou lokalizaci objektů neuronových sítí na problém, který je řešitelný jednoduchým výpočtem v rámci neuronové sítě ze vstupních informací. Tímto způsobem je dosaženo potřebné rychlosti pro lokalizaci objektu v živém přenosu.

Přestože celkové výsledky potvrdili schopnost algoritmu rozeznávat a vykreslovat polohu objektů s přijatelnou přesností, znatelně záleželo na vhodných podmínkách vstupního obrazu. Příčinou tohoto problému se jeví příliš malá množina učících obrazů, která není dostatečně rozmanitá pro kvalitní naučení neuronové sítě k lokalizaci objektů. Tento problém je ovšem řešitelný přípravou větší množiny dat, bez nutnosti opravovat připravený algoritmus nebo strukturu neuronové sítě. Překvapivě malý vliv měly na celkovou funkcionalitu úpravy parametrů neuronové sítě jako velikost skryté vrstvy či počet iterací učících cyklů.

Pro uvedenou práci existuje několik způsobů rozšíření, které by vedli ke zlepšení funkcionality celého programu. Kromě rozsáhlejší množiny učících dat to je především schopnost detekovat více objektů najednou, popřípadě tvorba projektu plně nezávislého na cizím prostředí.

6 Seznam použitých materiálů

1. **Xavier Glorot, Antoine Border, Yoshua Bengio.** Deep Sparse Rectifier Neural Networks. Université de Montréal.

2. **Dominik Scherer, Andreas Muller, Svne Behnke.** *Evaluation of Pooling Operations in Convolutiona Architectures for Object Recognition.* University of Bonn, Institute of Computer Science, 2010.

3. **Britz, Denny.** Understanding Convolutional Neural Networks for NLP. *wildml.* [Online] 7. 10 2015. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

4. **Job Tobias Springerberg, Alexey Dosovitskiy, Thomas Borx, Martin Riedmiller.** *Striving for Simplicity: The All Convolutional Net.* University of Freiburg, 2015.

5. **Benenson, Rodrigo.** Classification datasets results. *Are we there yet ?* [Online] 2013-2016. http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.

6. **K.E.A van de Sande, T. Gevers, A.W.M. Smeulders.** Selective Search for Object Recognition. University of Trento, University of Amsterdam, 2012.

7. **Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun.** *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* 2016.

8. **Pierre Sermanet, David Eigen,Xiang Zhang,Michael Mathieu,Rob Fergus,Yann LeCun.** *OverFeat:Integrated Recognition, Localization and Detection using Convolutional Networks.* New York : Courant Institute of Mathematical Sciences, New York University, 2014.

9. **Yangqing Jia, Evan Shelhamer.** Caffe. *Caffe.* [Online] The Berkeley Artificial Intelligence Research. http://caffe.berkeleyvision.org/.

10. **Skymind.** Introduction to Deep Neural Networks. *DL4J.* [Online] Skymind. https://deeplearning4j.org/index.html.

11. **Chalupník, Vitalij.** Biologicky inspirované algoritmy. *Root.* [Online] https://www.root.cz/serialy/biologicky-inspirovane-algoritmy/.

12. **Ross Girshick, Jeff Donahue,Trevor Darrell,Jitendra Malik.** *Rich feature hierarchies for accurate object detection and semantic segmentation.* 2014.

13. **Everingham M., Van~Gool L., Williams C.K.I., Winn J., Zisserman A.** Visual Object Classes Challenge 2012 Dataset. [Online] http://academictorrents.com/details/df0aad374e63b3214ef9e92e178580ce27570e59.

14. Performing Convolution Operations. [Online] Apple Inc. https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html.

15. **Genevieve Orr, Nici Schraudolph, Fred Cummins.** Neural Networks. *www.willamette.edu.* [Online] Willamette University. https://www.willamette.edu/~gorr/classes/cs449/intro.html.

16. **A. Rozantsev, V. Lepetit, P. Fua.** Detecting Flying Objects using a Single Moving Camera, accepted in IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016

17. **K. R. Sapkota, S. A. Roelofsen, A. Rozantsev, V. Lepetit, D. Gillet et al.** Vision-Based Unmanned Aerial Vehicle Detection and Tracking for Sense and Avoid Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems, Daejeon, Korea, 2016

7 Seznam tabulek

8 Seznam obrázků

9 Obsah přiloženého CD

K této práci jsou přiloženy skripty, které sloužily pro vytvoření dat, skripty k otestování funkcionality sítě a soubory potřebné k jejich správnému zprovoznění. Dále je na CD přiložena kopie práce v PDF:

```
/
-Lokalizace
    --Lokalizace_z_videa.py
    --Lokalizace_z_kamery.py
    --Podpůrné skripty pro tvorbu dat
        ---rozsekani_textu.m
        ---image_augmentation.m
        ---hashing.m
        ---unhashing.m
        ---image_labeling.m
        ---data_download.java
    --database_testovacich_obrazu
        ---database.lmdb
            ----data.lmdb
            ----lock.lmdb
    --database_ucicich_obrazu
        ---database.lmdb
            ----data.lmdb
            ----lock.lmdb
    --odkaz_k_testovacim_datum.txt
    --IoULayer.py
    --README_instalace_a_spusteni.txt
    --Solver_Site.prototxt
    --Architektura_site.prototxt
    --odkaz_k_datum.txt
    --naucene_vahy.caffemodel
    --testovaci_obrazky_souradnice_polohy.h5
    --ucici_obrazky_souradnice_polohy.h5
    --testovaci_video.mov
-dp_2017_Ficenec_Adam.pdf
```