VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKTOR HLAVY V OBRAZE

DETECTOR OF THE HUMAN HEAD IN IMAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB SVOBODA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ GOLDMANN**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Svoboda Jakub**
Obor: Informační technologie
Téma: **Detektor hlavy v obraze**
Detector of the Human Head in Image
Kategorie: Zpracování obrazu

Pokyny:

1. Seznamte se s existujícími detektory hlavy v obraze. Především se zaměřte na řešení využívající neuronové sítě.
2. Nastudujte nejčastěji používané typy neuronových sítí k detekci objektů v obraze.
3. Navrhněte detektor hlavy založený na neuronové síti, který bude detekovat hlavu v různých pozicích (nejen podle obličeje).
4. Navržený detektor implementujte jako multiplatformní desktopovou aplikaci.
5. Proveďte experimenty a určete na základě biometrických metrik výkonnost detektoru.

Literatura:

- SZEGEDY, Christian, et al. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015. p. 1-9.
- VU, Tuan-Hung; OSOKIN, Anton; LAPTEV, Ivan. Context-aware CNNs for person head detection. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015. p. 2893-2901.
- 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.

Pro udělení zápočtu za první semestr je požadováno:

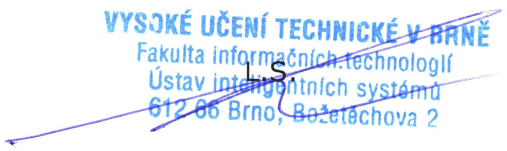
- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Goldmann Tomáš, Ing.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018



doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Detekce hlavy je důležitou součástí algoritmů pro detekci a identifikaci osob. Tato práce je zaměřena na detekci lidské hlavy v obraze pomocí neuronových sítí. Většina konvenčních detektorů dokáže detekovat objekty v omezené množině úhlů, zatímco modely založené na neuronových sítích pokrývají větší škálu úhlů natočení hlavy. V této práci jsme natrénovali současné *state-of-the-art* modely a porovnali je z hlediska přesnosti a rychlosti zpracování snímku. Nejvíce se osvědčil model RetinaNet, který dosáhl přesnosti 85,15 % AP. Díky tomuto detektoru mohou být vylepšené dostupné algoritmy pro detekci, identifikaci a sledování osob.

Abstract

Detection of human head is an important part of person detection and identification algorithms. This thesis is focused on the detection of human head with methods based on neural networks. The majority the of conventional detectors can identify objects within a limited range of positions, whereas models based on neural networks offer a more robust approach. In this thesis we trained the current *state-of-the-art* models and compared their accuracy and speed. The most accurate model proved to be RetinaNet which has reached 85.15 % AP. This detector can be used to improve current available algorithms for person detection, identification and tracking.

Klíčová slova

detekce objektů, strojové učení, konvoluční neuronové sítě, hluboké neuronové sítě, Faster R-CNN, YOLO, You Only Look Once, SSD, Single Shot MultiBox Detector, Mask RCNN, RetinaNet

Keywords

object detection, machine learning, convolutional neural networks, deep neural networks, Faster R-CNN, YOLO, You Only Look Once, SSD, Single Shot Multibox Detector, Mask RCNN, RetinaNet

Citace

SVOBODA, Jakub. *Detektor hlavy v obraze*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Goldmann

Detektor hlavy v obraze

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Goldmanna. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Svoboda
7. května 2018

Poděkování

Rád bych poděkoval panu Ing. Tomášovi Goldmannovi za skvělé vedení a cenné rady, které mi poskytl při tvorbě této bakalářské práce. Dále bych chtěl poděkovat národnímu výpočetnímu centru IT4Innovations. Velmi oceňuji poskytnutí výpočetních a úložných zařízení, bez kterých by tato práce nevznikla. Především bych chtěl poděkovat panu Ing. Josefu Hrabalovi za doinstalování modulů potřebných pro tuto bakalářskou práci na cluster Anselm.

Obsah

Kapitola 1

Úvod

V posledních letech je zaznamenán velký nárůst množství zachycených obrazových dat. Tam, kde dříve stačilo tato data manuálně zpracovat, je v dnešní době nutné nalézt způsoby, jak zpracovaní obrazu zautomatizovat. Díky rychlému rozvoji především paralelního výpočetního výkonu se dnes upouští od zaběhlých algoritmů a stále častěji jsou využívány konvoluční neuronové sítě, které svojí přesností při rozpoznávání a lokalizaci objektů překonávají dosud používané metody [5]. Detekce lidské hlavy je jedním z těchto případů a touto úlohou se zabývá i tato práce.

Detekce osob nebo lidských hlav se dnes využívá na mnoha místech. Typicky se může jednat o kamery v bezpečnostních systémech na letištích, kde tyto systémy automaticky detekují počet osob v hlídaném prostoru, popřípadě mohou upozornit ostrahu na vstup osob do zakázaných míst. Využití lze najít i ve veřejném sektoru. Kamery umístěné v nákupním středisku mohou detekovat přítomnost osob a sledovat jejich pohyb prodejnou nebo zjišťovat zájem zákazníků o určitý druh zboží.

Cílem této práce bylo prostudovat možnosti detekce a následné lokalizace lidské hlavy v obraze v různých pozicích, navrhnout řešení tohoto problému založeném na detektoru využívajícím konvoluční neuronové sítě, natrénovat samotný detekční model a na základě biometrických metrik pak určit výkonnost detektoru.

V kapitole 2 je čtenáři blíže popsána problematika počítačového vidění. V podkapitole 2.1 jsou pak podrobněji vysvětleny detekční algoritmy Viola and Jones a algoritmus využívající histogramy orientovaných gradientů (HOG). Závěr kapitoly je věnován neuronovým sítím, je zde popsán model umělého neuronu a principy učení neuronových sítí. Jsou zde také rozebrány nejpoužívanější chybové a aktivační funkce. Ke konci kapitoly jsou popsány samotné vrstvy neuronových sítí a proces konvoluce.

Kapitola 3 se věnuje současným *state-of-the-art* detekčním sítím. Podrobněji je zde popsán princip fungování pěti, v dnešní době nejpoužívanějších, detektorů. V kapitole 4 jsou popsány současné volně dostupné datasety, které lze pro trénování modelů využít. V této kapitole je pak popsáno trénování vybraných modelů pro využití při detekci lidské hlavy a implementace multiplatformní aplikace založené na detektoru RetinaNet. U jednoho modelu je pak zkoumána možnost dotrénování detektoru pro scénu se statickou kamerou z anotací vytvořených samotným detektorem.

V kapitole 5 je vyhodnocena úspěšnost jednotlivých modelů detekovat lidskou hlavu nad testovacími daty, v podkapitole 5.3 jsou pak zobrazeny detekce na ukázkových fotografiích a detekce jednotlivých modelů jsou porovnány. Následně je zde porovnána úspěšnost modelu

natrénovaného na obecných datech s modelem dotrénovaným z vlastních anotací ve scéně se statickou kamerou. V této kapitole je také porovnána rychlost detektorů a to zvlášť při běhu na procesoru a při akceleraci paralelních výpočtů pomocí grafické karty. V závěrečné, 6. kapitole je pak shrnuta provedená práce a je zde diskutován budoucí vývoj a možná rozšíření.

Kapitola 2

Metody pro detekci objektů v obraze

Samotný problém detekce objektů v obraze lze rozdělit na několik dílčích kroků. Nejprve je nutné získat samotná vstupní obrazová data. Nejčastějším zdrojem jsou fotoaparáty nebo videokamery, které pomocí technologie CCD nebo CMOS zachycují informace v digitální podobě. V dnešní době se stále častěji setkáváme s využitím počítačového vidění pro lékařské účely a zdrojem obrazových dat tak může být i magnetická rezonance nebo výpočetní tomografie.

Získaná obrazová data jsou následně předzpracována. Může se jednat o jednoduché úpravy, jako je změna rozlišení, ale také o zvýraznění hran objektů, úpravu jasnosti fotografie nebo normalizaci barev a kontrastu. Jednotlivé úpravy se liší podle použité metody detekce, někdy může být předzpracování obrazu vynecháno úplně.

Dalším krokem je segmentace obrazu, kdy je cílem lokalizovat místa s možným výskytem hledaného objektu (*Region of Interest* – ROI). Nejjednodušší metody pracují s oknem, které se po určitém kroku (*stride*) posouvá po obraze a výřez z tohoto okna je poté předán klasifikátoru. Tuto metodu využívá například metoda Viola Jones, která je blíže popsána v podkapitole 2.1.1. Problémy tohoto algoritmu vyvstávají z příliš velkého počtu výřezů, které je nutné zpracovat. Stejně tak pro detekci objektů s různým poměrem stran a orientací je nutné použít okno opakovaně s různými parametry, což je výpočetně náročné. V dnešní době využívá valná většina detektorů k nalezení objektů neuronové sítě, které jsou schopny selektivně vybírat vhodné hraniční oblasti a snížit tak celkový počet výřezů, které jsou předány klasifikátoru.

Dalším krokem při detekci objektů je extrakce příznaků. Příznaky mohou být například barva, struktura nebo tvar objektu. Tyto příznaky slouží jako popis samotného objektu. U neuronových sítí je extrakce příznaků prováděna pomocí konvolučních vrstev. Získané příznaky jsou předány klasifikátoru, který rozhodne, zda se jedná o objekt z jedné z předem známých kategorií (*image classification*). V následující podkapitole je popsáno, jak k tomuto problému přistupují jednotlivé algoritmy. Zvláštní pozornost je věnována detekčním algoritmům založeným na neuronových sítích, které svoji přesností překonávají tradiční algoritmy [5].

2.1 Algoritmy detekce objektů

Prvotní metody detekce byly založené na principu porovnávání tvaru nebo barvy a nepřinášely dostatečně přesné výsledky pro reálné využití. Až s příchodem složitějších algoritmů, jako je například Eigenfaces [31] nebo metoda Schneiderman-Kanade, bylo možné prakticky využít detektory v omezených podmínkách. K výraznému rozvoji došlo až po přelomu tisíciletí, mimo jiné také díky pokroku ve vývoji hardware a tím pádem dostupnějšímu výpočetnímu výkonu.

2.1.1 Algoritmus Viola a Jones

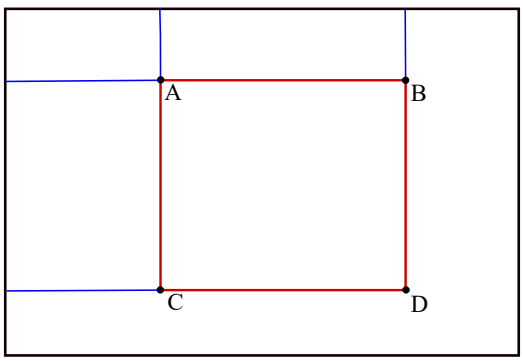
Jeden z prvních detektorů představili Paul Viola a Michael Jones v roce 2001 [33]. Jimi navrhnutý algoritmus, využívající Haarovy příznaky, je možné využít na detekci objektů. Demonstrační aplikace, kterou v roce 2001 představili, byla schopná v téměř reálném čase zpracovávat video z webové kamery a v obraze detekovat obličeje kolemjdoucích.

Haarovy příznaky mají tvar obdélníků s různou orientací, obvykle značených bílou a černou barvou. Hodnota těchto příznaků se spočítá jako rozdíl sumy intenzity pixelů (ve stupních šedi) pod těmito dvěma částmi. Haarovy příznaky vycházejí z Haarových vlnek (popsány Haarovou funkcí 2.1). Kvůli vysokému počtu příznaků, a z toho vyplývající výpočetní náročnosti, není samotný detektor tvořen všemi příznaky, ale pouze těmi, které nejvíce přispívají k rozpoznání obrazu. K tomu je využit algoritmus AdaBoost.

Pro zefektivnění práce s příznaky navrhli Viola a Jones postup, kdy je původní obraz převeden do integrálního obrazu. Na místě původních pixelů je uchovávána hodnota, která značí součet intenzit všech pixelů, které se nachází vlevo nahoře od tohoto pixelu. Reprezentovat obraz tímto způsobem je výhodné, protože při zjišťování intenzity v obdélníkové zóně v obrazu lze pracovat pouze se čtyřmi okrajovými body zóny a není nutné oblast procházet přes všechny pixely.

$$\psi(x) = \begin{cases} 1 & 0 < t < \frac{1}{2} \\ -1 & \frac{1}{2} < t < 1 \\ 0 & jinde \end{cases} \tag{2.1}$$

Předpis Haarovy funkce [36]



Obrázek 2.1: Znázornění výpočtu plochy s integrálním obrazem pomocí vztahu $\sum = D - B - C + A$.

2.1.2 Histogram of Oriented Gradients (HOG)

Detektor založený na histogramech orientovaných gradientů (HOG) je metoda detekce objektů v obraze. Algoritmus byl poprvé představen již roku 1986 [19], ale jeho širší využití nastalo až v roce 2005, kdy byla tato metoda použita na detekci chodců ve videu a dosahovala výrazně lepších výsledků než ostatní, do té doby dostupná řešení. HOG detektor je implementován v běžně dostupných knihovnách, např. OpenCV nebo Matlab.

Před samotným výpočtem je možné obraz předzpracovat normalizací kontrastu a barev. Dalal a Triggs však ve své práci [1] ukázali, že normalizace sice vedla k přesnějším výsledkům, ale zlepšení bylo minimální. Mnoho implementací HOG detektoru proto předzpracovaní zcela vynechává.

Prvním krokem modelu je výpočet horizontálního a vertikálního gradientu G pro každý bod obrazu na souřadnicích x a y pomocí vztahu

$$G = \begin{pmatrix} G_x \\ G_y \end{pmatrix} = \begin{pmatrix} \frac{df}{dx} \\ \frac{df}{dy} \end{pmatrix}. \tag{2.2}$$

Gradient reprezentuje lokální změnu intenzity mezi sousedními pixely. Výsledná hodnota bude tedy vyšší v místech, kde se v obraze nacházejí hrany objektů a kde je tím pádem barevná změna nejvyšší. Výpočet gradientu umožňuje odstranit z obrazu informace, které nutně nepřispívají k rozpoznání objektu. U barevných fotografií je gradient vypočten zvlášť pro každý kanál a pro daný pixel je zvolen ten s nejvyšší hodnotou. U gradientu G je následně vypočtena jeho velikost $|G|$ a směr θ:

$$|G| = \sqrt{G_x^2 + G_y^2}, \tag{2.3}$$

$$\theta = \arctan\left(\frac{G_x}{G_y}\right). \tag{2.4}$$

Dalším krokem je rozdělení obrazu mřížkou do buněk. Pro každou buňku je určen histogram, jehož kanály reprezentují zastoupení směru jednotlivých gradientů z předchozího kroku. Dalal a Triggs empiricky dokázali, že rozdělení histogramu na devět kanálů přineslo nejpřesnější výsledky. Některé implementace algoritmu využívají rozsah orientace gradientu 0 až 180 stupňů, každý kanál histogramu pak odpovídá úhlům v rozsahu 20 stupňů. Empiricky však bylo dokázáno [1], že při použití histogramu úhlů v rozsahu 0 až 360 stupňů lze dosáhnout lepších výsledků.

Histogramy z předchozího kroku jsou závislé na celkové světelnosti fotografie. Pro správnou funkčnost je nutné, aby model byl nezávislý na světelnosti a podával dobré výsledky i u tmavých nebo přesvětlených fotografií. Histogram je tedy normalizován a to po větších blocích, obvykle po čtyřech sousedních buňkách [1]. Autoři otestovali několik normalizačních metod a empiricky zjistili, že metody L2-Hys, L2-norm a L1-sqrt byly srovnatelně účinné, zatímco metoda L1-norm negativně ovlivnila úspěšnost detektoru. Vynechání normalizace pak mělo za následek zhoršení detekcí o 27 %. Z normalizovaných histogramů je následně konkatenací sestaven jeden vektor, který je předán klasifikátoru. Ten rozhodne, jestli se hledaný objekt v obraze nachází.

$$\mathrm{L1-norm} = \frac{v}{||v||_1 + e} \tag{2.5}$$

$$\mathrm{L1-sqrt} = \sqrt{\frac{v}{||v||_1 + e}} \tag{2.6}$$

$$\mathrm{L2-norm} = \frac{v}{\sqrt{||v||_2^2 + e^2}} \tag{2.7}$$

Rovnice 2.5 až 2.7 představují metody pro normalizaci světelnosti obrazu [1], kde v značí nenormalizovaný vektor skládající se ze všech histogramů pro daný blok, e je konstantní hodnota.

2.2 Neuronové sítě

Umělé neuronové sítě jsou výpočetní struktury používané v oblasti umělé inteligence. Jejich vzorem jsou sítě mozkových buněk – neuronů v živých organismech. Neuronové sítě jsou schopny se naučit plnit zadaný úkol, aniž by na daný problém byly specificky programovány. Vytrénované neuronové sítě jsou schopny odhalit vzorce z velkého množství dat, mnohdy s větší úspěšností než lidé [20].

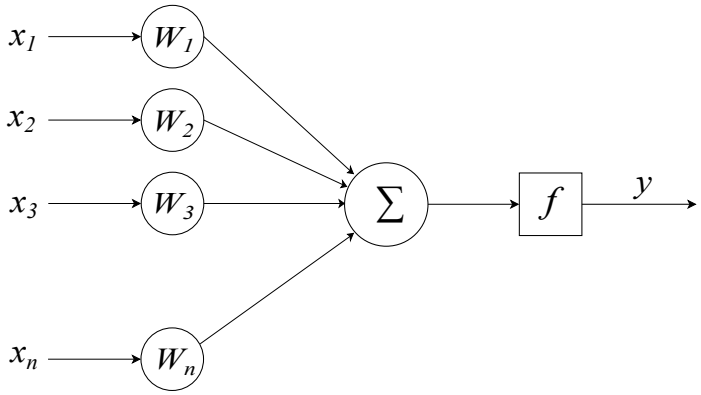
Úkony, prováděné neuronovými sítěmi, lze obecně klasifikovat na učení s učitelem (*supervised learning*) a učení bez učitele (*unsupervised learning*). Při učení s učitelem má síť k dispozici zpětnou vazbu, která indikuje, kdy systém pracuje správně a může také poskytovat informace o velikosti chyby [20]. Příkladem takového učení může být rozpoznávání obsahu obrazu, kdy se síť učí na předem označených (*labeled*) datech. Oproti tomu učení bez učitele spoléhá na heuristicky získané informace a systém sám vyhledává a rozpoznává vzorce v datech. Typickým problém řešeným učením bez učitele je shlukování (*clustering*).

2.2.1 Model neuronu

Základními stavebními prvky neuronových sítí jsou neurony. Umělý neuron (viz obrázek 2.2) je funkce s vektorem vstupů (ekvivalent dendritů v lidském mozku) a vektorem vah. Při výpočtu jsou váhy a vstupní hodnoty navzájem vynásobeny. K sumě těchto hodnot je pak přičtena skalární hodnota – *bias* a výsledek je předán aktivační funkci, po jejímž výpočtu je výsledná hodnota předána z neuronu dále, obvykle do další vrstvy neuronů. Pokud se jedná o poslední vrstvu, je vyhodnocen výsledek. [20].

2.2.2 Učení neuronových sítí

Pro učení fungování neuronové sítě je nutná existence algoritmu, který provede změnu vah a biasů tak, aby při vstupu vzorku dat z datasetu do sítě odpovídal výstup očekávanému výsledku. Ke zjištění toho, do jaké míry se výstup sítě liší od ideálního výsledku očekávaného u trénovacích dat, se využívá chybových funkcí. Z jejich výstupu (*loss*) je při trénování určeno, jakým směrem a do jaké míry je nutné změnit váhové hodnoty neuronů. Správný průběh trénování neuronové sítě lze tak pozorovat z hodnoty chybové funkce, která by se měla po každé iteraci snížit a síť by tak měla stále lépe aproximovat funkci specifickou pro daný úkol.



Obrázek 2.2: Model umělého neuronu s vektorem vstupů $x = (x_1, x_2, x_3, \ldots, x_n)$. Každá vstupní hodnota je vynásobena váhovým koeficientem $W = (W_1, W_2, W_3, \ldots, W_n)$. Suma těchto hodnot je pak předána aktivační funkci f, která vypočítá výstupní hodnotu y.

Chybové funkce

Mean squared error (MSE) je funkce pro výpočet chyby. Umocnění slouží k odstranění záporných hodnot a zároveň způsobí zvýraznění větších chyb. MSE je definována jako

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2, \tag{2.8}$$

kde $\hat{Y}_i$ je vektor predikcí, n je velikost tohoto vektoru a Y_i je vektor očekávaných výsledků (*ground-truth*).

Cross entrophy loss je chybová funkce, která se využívá v případech, kdy výstup může nabývat hodnot z několika tříd, které nejsou vzájemně výlučné. V praxi se může jednat například o rozpoznání objektu v obraze. Funkce je definována vzorcem

$$\mathrm{CE} = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k}\ln(y(x_n, w)), \tag{2.9}$$

kde x značí vstupní vektor o velikosti N, $y(x_n, w)$ je výstup z neuronové sítě za použití vah w a t_n je očekávaný výstup.

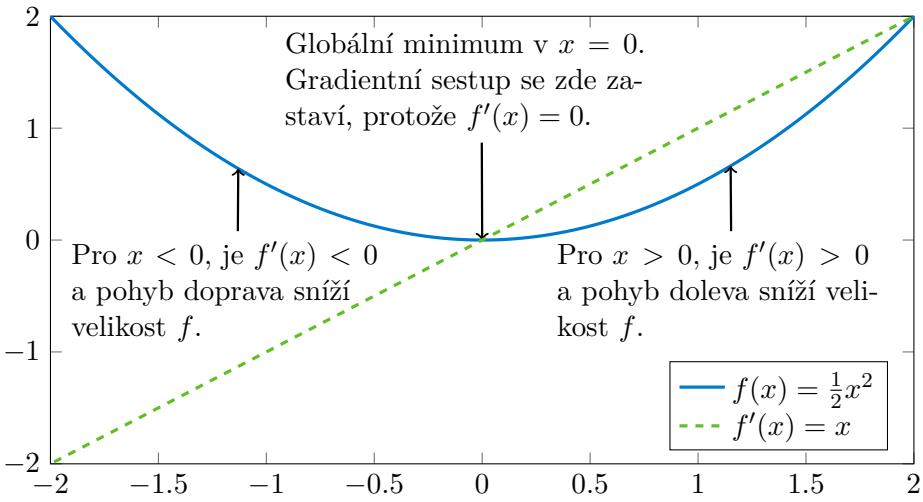
Gradient descent je iterativní, optimalizační algoritmus pro hledání minima funkce. Chyba funkce se zvětšuje ve směru nejvyššího gradientu (∇G), při hledání minima tedy algoritmus postupuje po krocích proti směru největšího gradientu. Velikost kroku se zmenšuje proporcionálně s velikostí gradientu. Pro změnu vah v neuronech se používá předpis

$$w_{i+1} = w_i - \eta\nabla G(w_i), \tag{2.10}$$

kde w značí váhu, η je velikost učícího kroku (*learning rate*), udávající rychlost změny vah. ∇G je gradient pro aktuální iteraci gradiálního sestupu. Algoritmus je ukončen v momentě, kdy je změna vah menší, než požadovaná hodnota.

Stochaický gradientní sestup (SGD) využívá pro změnu vah pouze jeden nebo několik vzorků dat (*mini-batch*), což má za následek snížení výpočetní náročnosti. Stochaický

gradient je spolu se zpětným šířením chyby nejpoužívanějším algoritmem pro trénování neuronových sítí [20].



Obrázek 2.3: Ukázka algoritmu Gradient Descent při hledání minima funkce [11].

Inicializace vah

Inicializace vah je proces, kdy je na začátku trénování neuronové sítě každému vstupu do neuronu přiřazen určitý koeficient (*weight*) a od těchto hodnot se odvíjí samotné učení. Vhodná inicializace vah může zásadně ovlivnit rychlost učení a pomoci ke konvergenci. Jednou z překážek pro široké využití hlubokých neuronových sítí je fakt, že trénování sítí s více než pěti vrstvami za pomocí algoritmu back-propagation je velmi obtížné a to zejména s nesprávně inicializovanými vahami [21].

Pro inicializaci vah a biasů je možné použít existující hodnoty ze sítě, která již byla natrénována. Tento způsob inicializace se nazývá *fine-tuning* nebo také *transfer learning* a je doporučený pro případy, kdy byly vrstvy předtrénované na podobných datech, jako obsahuje cílový dataset [21]. Obecně platí, že první vrstvy u detekčních neuronových sítí rozpoznávají základní tvary a hrany objektů, zatímco konečná identifikace objektu nastává až u posledních vrstev. Je tak vhodné opakovaně použít váhy z prvních vrstev, u kterých lze předpokládat, že jejich hodnoty se pro detekci různých objektů nebudou výrazně odlišovat.

Backpropagation

Backpropagation (zpětné šíření chyby) je adaptační algoritmus, pomocí kterého lze vypočítat podíl jednotlivých neuronů na celkové chybě a upravit váhy jednotlivých neuronů tak, aby byla minimalizována chyba. Jedná se o nejrozšířenější adaptační algoritmus, který je používaný v přibližně 80 % všech aplikací neuronových sítí [34].

Samotný algoritmus se skládá ze tří, neustále se opakujících fází. Nejprve je vstupní signál šířen sítí dopředu (*feedforward*), následně je chyba šířena v opačném směru a nakonec jsou aktualizovány jednotlivé váhy včetně biasů na spojeních neuronů [34]. Propagací chyby zpět do sítě dochází ke změně vah neuronů proti směru gradientu tak, aby se chybová funkce blížila minimu. Parametry sítě se upraví v závislosti na učícím kroku (*learning rate*)

a velikosti gradientu, čímž se síť trénuje. Velikost učícího kroku se může v průběhu procesu učení měnit, například skokovým zmenšením po dosažení určitého počtu iterací.

Pro výpočet parciálních derivací se využívá řetězové pravidlo (*chain rule*) [16]:

$$\begin{aligned} \Delta z &= \frac{\partial z}{\partial y}\Delta y \\ \Delta y &= \frac{\partial y}{\partial x}\Delta x \\ \Delta z &= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x \\ \frac{\partial z}{\partial x} &= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}. \end{aligned} \tag{2.11}$$

2.2.3 Aktivační funkce

Aktivační funkce neuronu slouží pro výpočet výstupní hodnoty pro daný vektor vstupních hodnot. Na rozdíl od jednoduššího perceptronu nemusí být výstupem binární hodnota. Volbou vhodné aktivační funkce lze významně urychlit výpočty při trénování neuronových sítí a pomoci síti ke konvergenci. Hodnota vypočtená na každém neuronu je předána aktivační funkci. Celý proces tak lze vyjádřit vztahem

$$a_j = \sigma(w_{i,j} * x_i + b_j) \tag{2.12}$$

kde σ značí chybovou funkci, w_i jsou váhové koeficienty, x_i jsou vstupní hodnoty do neuronu a b_j značí bias.

Prahová funkce

Prahová funkce je nejjednodušší přenosová funkce. Pokud součet vstupních hodnot vynásobený vahami pro jejich hrany přesáhne prahovou hodnotu, je neuron aktivován a na výstup je předána hodnota jedna. Binární výstup prahové funkce však není ve většině případů vhodný. Z tohoto důvodu není tato funkce v praxi příliš využívána. Prahová funkce je definována jako

$$f(x) = \begin{cases} 1 & \text{pro} \quad x \geq t \\ 0 & \text{pro} \quad x < t \end{cases} \tag{2.13}$$

kde x je vstupní hodnota prahové funkce a t je prahová hodnota (*threshold*).

Lineární funkce

Lineární funkce je přenosová funkce, u které je výstup proporcionálně roven vstupům, tedy váženému součtu hodnot na vstupu do neuronu. Možným výstupem funkce je rozsah hodnot, nejedná se tedy o binární aktivaci, jako tomu bylo u prahové funkce a na lineární aktivační funkci se tak nevztahují problémy spojené s binárním výstupem. Funkce je definována jednoduchým vztahem, kde parametr x je váženým součtem hodnot na vstupu do neuronu

$$f(x) = x. \tag{2.14}$$

Problém u lineární přechodové funkce však vyvstává při výpočtu derivace podle x, která je konstantní a nezávisí tedy na hodnotě x, což způsobuje problém při výpočtu gradientního sestupu.

Druhý problém při využití lineární funkce se projevuje u hlubokých neuronových sítí, kdy je každá vrstva aktivována lineární přechodovou funkcí. Výsledek po aktivaci se přenáší do další vrstvy a následné vrstvy s těmito hodnotami dále pracují v aktivačních vrstvách svých neuronů. To má za následek, že poslední aktivační funkce bude pouze lineární funkcí vstupu první vrstvy a všechny vrstvy by tedy bylo možné nahradit pouze jednou. Z tohoto důvodu je její využití spíše okrajové.

Sigmoidní funkce

Sigmoid je aktivační funkce s nebinárním výstupem z intervalu $(0, 1)$. Funkce a její kombinace jsou nelineární. Tyto vlastnosti činí sigmoid vhodnou funkcí pro využití v hlubokých neuronových sítích. Sigmoid lze definovat předpisem

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.15}$$

Pro vstupní hodnoty z intervalu $(-2, 2)$ strmě roste hodnota na ose y, což znamená, že funkce tíhne ke stranám křivky a je proto vhodná pro využití u klasifikace. Na tuto vlastnost se váže i několik problémů. Derivace se v bodech více vzdálených od středu osy x blíží nule a síť se z takto malých hodnot derivace nedokáže v rozumném čase učit. Tento problém je známý jako *vanishing gradient* [9] a je řešitelný buď úpravou výpočtu [22] nebo jednoduše použitím rychlejšího hardware.

Hyperbolický tangens

Průběh hyperbolického tangentu (*tanh*) je velmi podobný sigmoidě a potýká se se stejnými problémy (vanishing gradient). Na rozdíl od sigmoidy je výstup této funkce v intervalu $(-1, 1)$ a je vycentrovaný okolo nuly, což ulehčuje optimalizaci. Gradient této funkce je kolem středních hodnot vyšší než u sigmoidy.

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}. \tag{2.16}$$

ReLU

Rectified Linear Unit je aktivační funkce, jejíž výstup je roven x pro kladné hodnoty a nule pro záporné. Samotná funkce i její kombinace jsou nelineární. ReLU je navíc oproti sigmoidu nebo hyperbolickému tangentu méně výpočetně náročná, při náhodně inicializovaných vahách by v síti bylo 50 % neuronů, které by se díky záporné hodnotě neaktivovaly, stejně tak samotné matematické operace jsou jednodušší. Ačkoliv je chování biologického neuronu blíže sigmoidu [37], v umělých neuronových sítích je ReLU, díky úspoře času potřebného na výpočet, využívána častěji než sigmoid nebo hyperbolický tangent. Funkce ReLU je definována předpisem

$$f(x) = \begin{cases} 0 & \text{pro} \quad x < 0 \\ x & \text{pro} \quad x \geq 0. \end{cases} \tag{2.17}$$

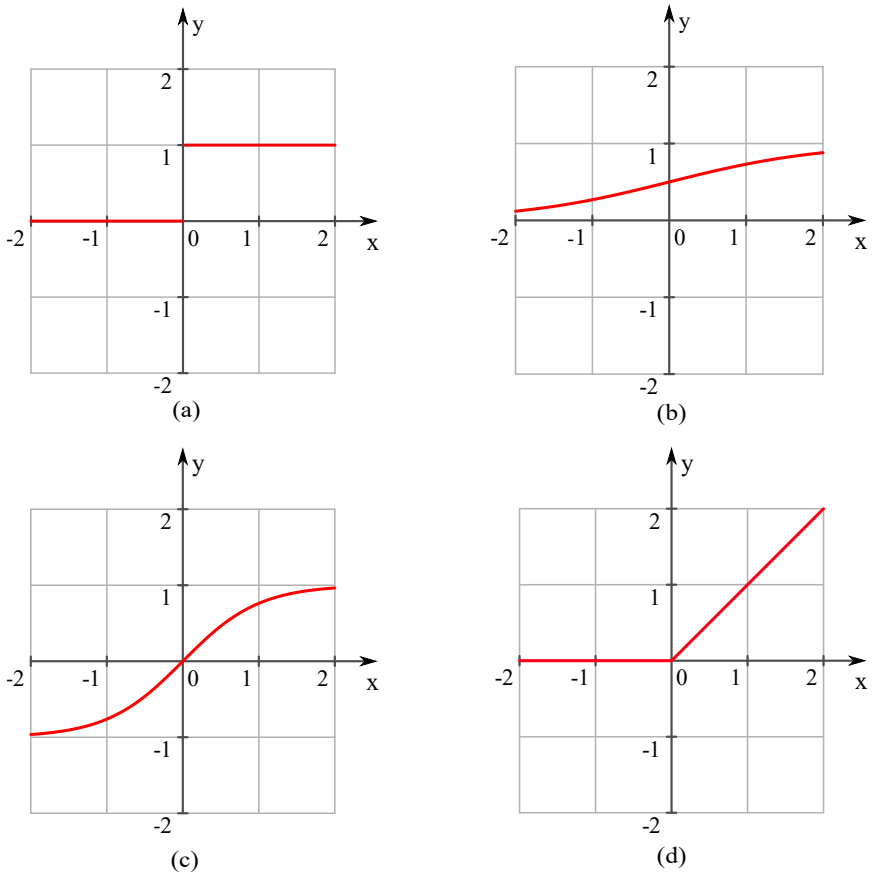
Pro záporné hodnoty x je gradient funkce ReLU nulový, což má za následek, že neuron může při učení přestat reagovat na podněty (tzv. *Dying ReLU problem* [37]). V praxi se proto často využívá obměna funkce ReLU, pro kterou jsou výstupy pro záporné hodnoty pouze velmi blízké nule (např. $y = 0{,}01x$ pro $x < 0$). Takto modifikovaná funkce se nazývá *Leaky ReLU*.

Softmax

Funkce softmax se využívá v posledních vrstvách neuronových sítí, obvykle tam, kde síť řeší klasifikační problém, například přiřazení objektu do předem definovaných tříd, které nejsou vzájemně výlučné. Obecně lze funkci softmax definovat jako

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{2.18}$$

pro $j = 1 \dots K$, kde funkce softmax normalizuje K rozměrný vektor z do K rozměrného vektoru $\sigma(z)$, jehož komponenty dávají v součtu hodnotu 1.

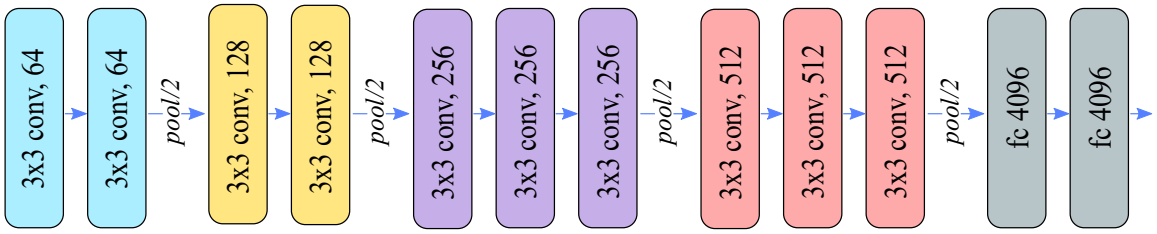


Obrázek 2.4: Grafy aktivačních funkcí. (a) Prahová funkce, (b) Sigmoida, (c) Hyperbolický tangent, (d) ReLU.

2.2.4 Vícevrstvé neuronové sítě

Vícevrstvé neuronové sítě jsou sítě skládající se ze dvou nebo více vrstev neuronů. První vrstva sítě přijímá vektorizovaná data a předává je do hlubších vrstev. Vrstvy mezi první a poslední vrstvou se nazývají skryté. Data, se kterými skryté vrstvy pracují, jsou směrem od vstupu více abstraktní a ke konci sítě lze pomocí příznaků definovat i velmi abstraktní datové struktury. Pokud má neuronová síť více než jednu skrytou vrstvu, hovoříme o ní jako o hluboké neuronové síti (*Deep Neural Network* - DNN).

Sítě pro zpracování obrazu využívají především konvoluční, plně propojené, pooling a dropout vrstvy. Ty jsou proto podrobněji popsány níže. Na obrázku 2.5 je zobrazené napojení jednotlivých vrstev v síti VGG16.



Obrázek 2.5: Schéma sítě VGG16 skládající se ze sérií konvolučních vrstev následovaných pooling vrstvou. Poslední dvě vrstvy jsou plně propojené.

Plně propojená vrstva (*fully connected layer*, někdy také *dense layer*) je vrstva neuronů, kde každý neuron přijímá hodnoty ze všech neuronů předchozí vrstvy, nebo ze vstupního vektoru dat. Sítě skládající se pouze z plně propojených vrstev se nazývají plně propojené neuronové sítě a jsou nejjednodušším příkladem neuronových sítí [20].

Ačkoliv jsou sítě s plně propojenými vrstvami schopny dobře aproximovat libovolnou funkci, vysoký počet spojení zvyšuje nároky na výpočetní kapacity. Proto jsou u detekčních sítí využívány zřídka a spíše se vyskytují v koncových vrstvách sítě, kde mají na starosti klasifikaci objektů.

Konvoluční vrstva je jedním z nejdůležitějších stavebních prvků konvolučních neuronových sítí. Její základní matematická operace je **konvoluce**. Tu lze popsat jako operaci nad dvěma funkcemi [16] a je definována jako

$$s(t) = (x \star w)(t) = \int x(a)w(t-a)da \tag{2.19}$$

kde $\star$ značí operaci konvoluce nad funkcemi $x(t)$ a $w(t)$. První argument konvoluce, v tomto případě funkce x, je obvykle označován jako vstup (*input*) a druhý argument, funkce w, jako konvoluční jádro (*kernel*).

Při použití ve zpracování obrazu jsou obvykle všechna data diskretizována, konvoluční vrstvy tedy pracují s diskrétní konvolucí. Ta je definována jako

$$s(t) = (x \star w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.20}$$

kde $\star$ značí operaci diskrétní konvoluce nad funkcemi $x(t)$ a $w(t)$.

Ve strojovém učení jsou vstupem a konvolučním jádrem často vícerozměrná pole, která se označují souhrnným názvem *tenzor*. Konvoluce je často uplatňována na více než jedné ose, například s dvourozměrným obrazem I jako vstupem s rozměry $w * h$ je použito i dvourozměrné konvoluční jádro K

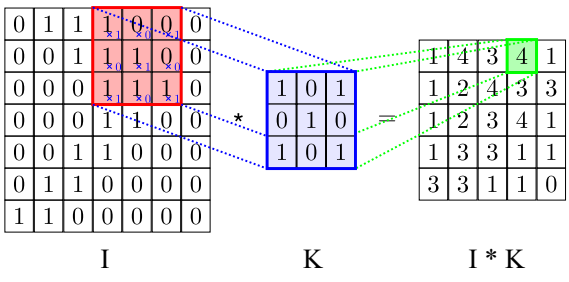
$$(I \star K)(x, y) = \sum_{i=1}^{h} \sum_{j=1}^{w} K(i, j) I(x + i - 1, y + j - 1). \tag{2.21}$$

Parametry konvoluční vrstvy se skládají z množiny konvolučních jader, která obvykle pracují se všemi barevnými kanály. Samotná síť se učí optimalizací parametrů těchto konvolučních jader (konvolučních filtrů). Filtry jsou po vstupu přesouvány po určitém kroku (*stride*) a na příslušných místech je spočítán skalární součin parametru a filtru. Je tak vytvořena dvoudimenzionální mapu, která zobrazuje místa, kde došlo k aktivaci [14]. Znázornění operace konvoluce je na obrázku 2.6.

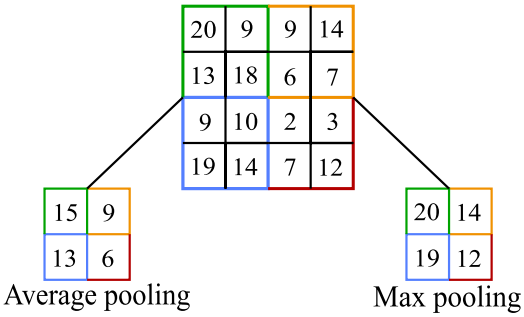
Výpočet konvoluce je problematický v okrajových částech obrazu. V těchto případech nemusí být konvoluce vypočítána tam, kde by konvoluční filtr přesahoval okraj vstupu (tzv. plná konvoluce). Tím je způsobeno zmenšení obrazu dané velikostí filtru. Alternativou je počítat konvoluci s nulovými body za hranicí obrazu (tzv. *zero padding*), popřípadě s jinou, vhodně vybranou hodnotou.

Pooling vrstva agreguje výstupy sousedních neuronů do jednoho, čímž dochází k podvzorkování a zmenšení rozměrů aktivační mapy. Tím lze výpočetně zjednodušit operace, které následují po této vrstvě. V praxi je běžné umístění pooling vrstvy vždy za konvoluční vrstvou [15]. Způsob agregace hodnot je například výpočet průměrné hodnoty, výběr minima nebo nejčastější výběr maxima (*max pooling*). Počet hodnot k agregaci je volitelný, je také možné využít překrývající pooling vrstvu, u které bylo empiricky dokázáno, že snižuje chybovost a náchylnost sítě k přetrénování [15]. Znázornění podvzorkování pomocí max pooling a average pooling je na obrázku 2.7.

Dropout vrstva se využívá ke snížení náchylnosti na přetrénování [15]. Pro každou skrytou jednotku nastaví na výstup hodnotu 0 s pravděpodobností 50 %, což způsobí, že aktivní skryté neurony nemohou spoléhat na přítomnost ostatních neuronů a ty jsou tak nuceny se učit. K deaktivaci neuronů dochází u každého vzorku dat při trénování a do učení je tak zanesen šum, díky čemuž je síť robustnější [8].



Obrázek 2.6: Konvoluce vstupu I a jádra K [32].



Obrázek 2.7: Ukázka operací *Average pooling* a *Max Pooling*.

Kapitola 3

Detekční sítě

Ačkoliv modely pro detekci objektů v obraze obvykle k samotné lokalizaci využívají rozdílné algoritmy, téměř všechny následně předávají výseče obrazu nějaké konvoluční síti, která klasifikuje danou část obrazu do předem známých kategorií. Úspěšnost lokalizace objektů je tak závislá na schopnosti sítě rozpoznat dané objekty s co nejmenším procentem chyb.

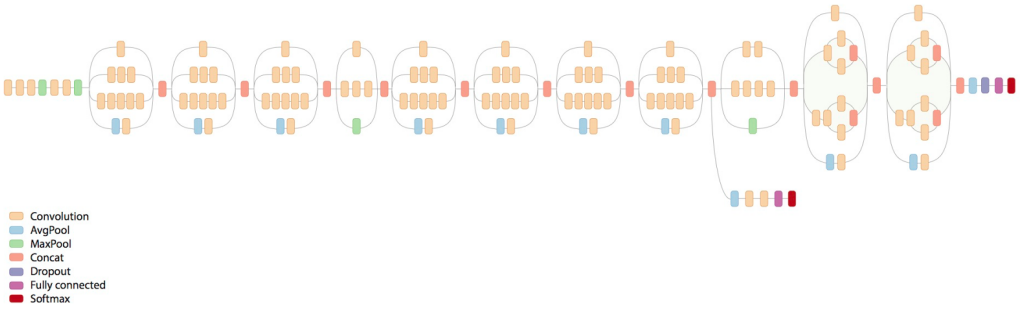
Konvoluční sítě byly pro rozpoznání obrazu využívány již od roku 1980 [3]. Novou vlnu zájmu o tuto oblast vzbudil v roce 2012 Alexandr Krizhevsky a jeho tým [15], který na soutěži ImageNet Large Scale Visual Recognition (ILSVR) se svojí sítí složenou z pěti konvolučních vrstev, tří pooling vrstev a tří plně propojených vrstev, zvítězil s výrazným náskokem. **AlexNet** dosáhl top-5 chybovosti 15,3 % (podíl případů, kdy model mezi pěti nejpravděpodobnějšími třídami neuvádí tu skutečnou), zatímco model, který se umístil na druhém místě, dosáhl pouze na 26,2 %. Za tímto výrazným zlepšením stojí především velký nárůst paralelního výpočetního výkonu, který umožnuje využívat neuronové sítě s více vrstvami a nárůst objemu označených dat pro trénování sítí [38]. Síť AlexNet využívá pro klasifikaci mimo jiné detektor R-CNN (sekce 3.1).

Model **ZF Net** [38] dosáhl top-5 chybovosti 11,2 % a zvítězil v ILRSV 2013. Oproti síti AlexNet, ze které vychází, byly pouze upraveny parametry filtru v první vrstvě (z 11×11 na 7×7) a zmenšen krok konvoluce. Sít využívá aktivační funkce ReLU a chybové funkce cross-entrophy loss. Zeiler a Fergus také přišli s metodou vizualizace příznaků pomocí dekonvolučních sítí, díky kterým je možné znázornit, které typy struktur aktivují danou mapu příznaků [38].

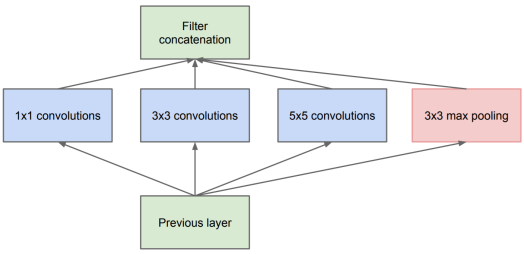
VGG Net z roku 2014 v konvolučních vrstvách využívá 3×3 filtru (tedy nejmenší velikost filtru, která pracuje s okolními pixely) s krokem o velikosti jedna [29]. Síť se skládá ze série konvolučních vrstev, které jsou vždy po dvou nebo třech vrstvách následovány max-pooling vrstvou. Před výstupem se ještě nacházejí dvě plně propojené vrstvy, viz obr. 2.5. Celkový počet vrstev se zvýšil až na 19 a s tímto počtem vrstev se top-5 chybovost pohybuje okolo 7,3 %.

V ILSVRC 2014 zvítězil model **GoogLeNet** (také známý jako Inception) s top-5 chybovostí 6,7 % [30]. Hlavním přínosem této sítě je zlepšení využití výpočetních zdrojů, což umožňuje vytvořit síť s hlubší a širší architekturou, než bylo běžné u předchozích sítí skládajících se ze sekvenčně seřazených vrstev. Model se skládal z na sebe navazujících modulů *inception*. V každém z těchto modulů jsou paralelně umístěny konvoluční vrstvy s filtry 1×1, 3×3 a 5×5. Schéma sítě Inception je zobrazeno na obrázku 3.1, modul inception pak na obrázcích 3.2 a 3.3.

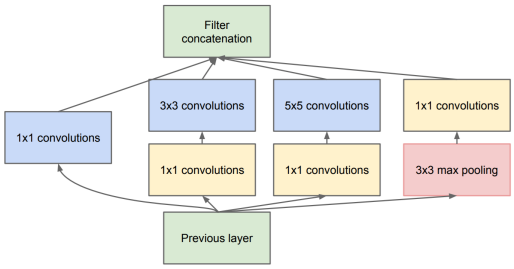
ResNet je síť od výzkumného týmu Microsoftu. S celkovým počtem 152 vrstev se jedná o jednu z nejrozsáhlejších architektur neuronových sítí. ResNet dosahuje chybovosti pouhých 3,57 % [7] (na soutěžním datasetu pro ILSVRC, ve které v roce 2015 zvítězil), což je vzhledem k faktu, že lidé dosahují chybovosti 5 – 10 % v závislosti na zkušenostech řešitele [13], výsledek velmi se blížící maximální teoreticky možné úspěšnosti.



Obrázek 3.1: Schéma sítě Inception V3 [28].



Obrázek 3.2: Naivní verze modulu Inception [30].



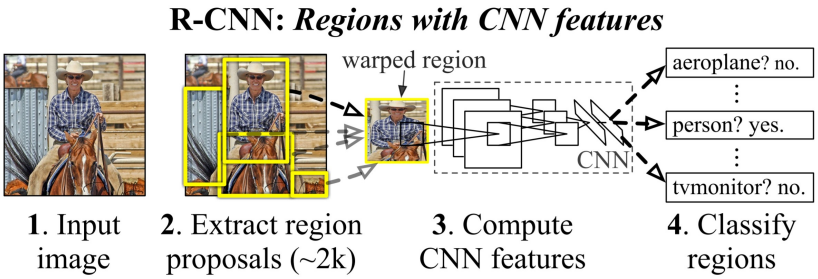
Obrázek 3.3: Úplné schéma modulu inception. 1×1 konvoluce zde slouží k redukci rozměrů [30].

3.1 Region-based Convolutional Neural Network

Region-based Convolutional Neural Network (RCNN) a detektory z něj vycházející byly významným pokrokem v oblasti detekce objektů [5]. Na datasetu ILSVRC2013 dosahoval model RCNN mAP 31,4 %, což je výrazné zlepšení oproti do té doby *state-of-the-art* modelu OverFeat [27] s mAP 24,3 % na stejném datasetu.

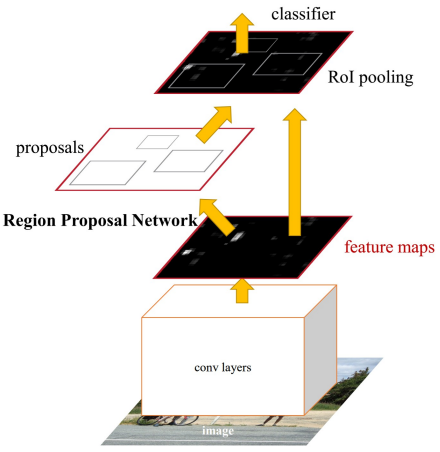
Autoři proces detekce rozdělili na dvě fáze. Nejprve model vyhledá možné výseče obrazu, kde se hledaný objekt pravděpodobně nachází a to nezávisle na kategorii, do které samotný objekt spadá. Těchto asi 2000 oblastí (*region proposals*) je nalezeno pomocí selektivního vyhledávání, kdy algoritmus prochází zdrojový obraz a snaží se spojit pixely, které jsou si barevně blízké a lze tak předpokládat, že náleží jednomu objektu. Oblasti jsou následně zdeformovány na vhodnou velikost a předány konvoluční neuronové sítí (autoři použili AlexNet). Po provedení konvolučních výpočtů je výsledek předán SVM (*Support*

Vector Machine), který rozhodne, jestli se ve výseči objekt skutečně nachází a do které z tříd patří. Celý proces je znázorněn na obrázku 3.4.



Obrázek 3.4: Schéma detekčního algoritmu R-CNN. Ze vstupního obrazu (1) jsou vybrány oblasti s možným výskytem objektů (2). Z těchto výsečí jsou pomocí konvoluční sítě (3) získány příznaky, které jsou předány SVM klasifikátoru (4) [5].

Původní model byl vylepšen především z důvodů neuspokojivé rychlosti detekce a nutnosti rozdělit trénování do několika nezávislých fází (extrakce příznaků pomocí AlexNet, SVM klasifikátor a regresor ohraničení), což je výpočetně velmi náročné. **Fast R-CNN** [4] tyto problémy řeší sdílením výpočtů konvolucí mezi jednotlivými okny s objekty (tzv. *Region of Interest Pooling*) a zároveň použitím pouze jedné sítě k extrakci příznaků a klasifikaci objektů (SVN klasifikátor nahrazen softmax vrstvou). Ke zlepšení výkonu také pomohlo použití sítě VGG16 oproti AlexNet.



Obrázek 3.5: Proces detekce pomocí Faster R-CNN. Vstupní obraz je zpracován konvoluční sítí a získané příznaky jsou předány RPN. Nad vybranými výsečemi je poté provedena klasifikace [26].

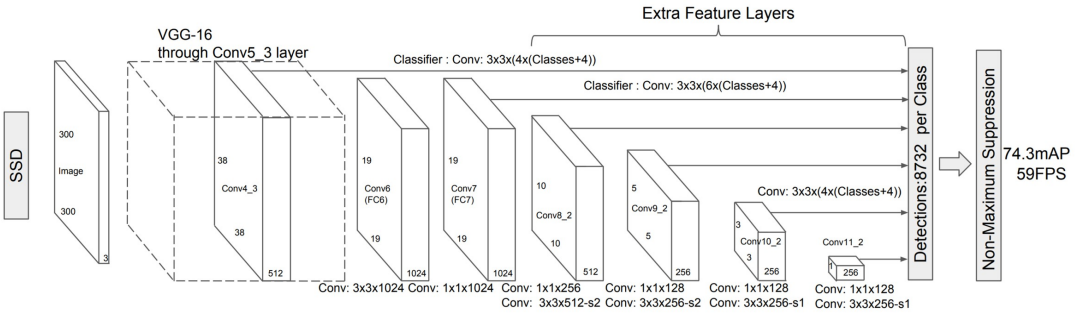
V roce 2016 byla představena nová iterace algoritmu – **Faster R-CNN** [26]. Autoři umístili za poslední konvoluční vrstvu síť RPN (*Region Proposal Network*). RPN z mapy příznaků z poslední konvoluční vrstvy vyhledá oblasti s možným výskytem objektu. Odtud je postup podobný jako u Fast R-CNN, následuje Region of Interest Pooling a klasifiace. Celý proces detekce je zobrazen na obrázku 3.5. Autoři detektoru uvádějí vyhledávání vhodných oblastí jako úzké hrdlo předchozích algoritmů a díky RPN byly schopni výrazně zvýšit efektivitu vyhledávání objektů (namísto 2000 oblastí u RCNN je jich u Faster R-CNN nutné

prohledat asi jen 300) a zároveň celý proces zrychlit, což dovoluje detekce téměř v reálném čase (autoři uvádějí okolo 5 snímků za sekundu na GPU [26]).

3.2 Single Shot Multibox Detector

Single Shot MultiBox Detector (**SSD**) [18] je model založený na dopředné konvoluční síti, která vytváří kolekci výsečí o dané velikosti a nad každou výsečí provede ohodnocení pravděpodobnosti, že se v ní nachází objekt spadající do určité třídy. Po ohodnocení je provedeno potlačení nemaxim (*non-maximum surpression*) podle určité mezní hodnoty, což způsobí vyfiltrování většiny výsečí.

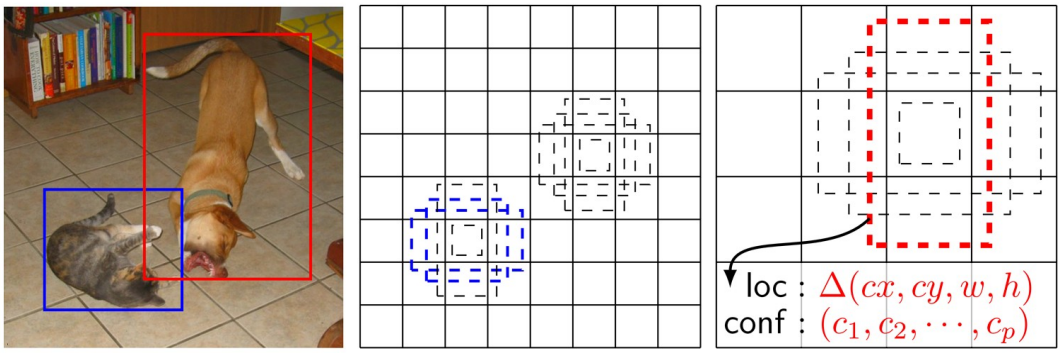
První vrstvy modelu jsou převzaty z konvoluční detekční sítě. Autoři algoritmu zvolili VGG16 (bez plně propojených vrstev), ale použití libovolné sítě s podobnou architekturou by mělo podávat dobré výsledky [18]. Za vrstvy původní sítě jsou přidány postupně se zmenšující konvoluční vrstvy, což pomáhá s detekcí objektů s různou velikostí. Na rozdíl od jiných detekčních algoritmů, které lokalizují objekty pouze na jedné vrstvě příznaků, provádí SSD detekce na různých vrstvách, což má za následek vyšší celkový počet nalezených objektů.



Obrázek 3.6: Schéma modelu SSD300. Vstupem do sítě je obraz o rozměrech 300×300 pixelů. Ten je předán konvoluční síti VGG16 (bez plně propojených vrstev), za kterou jsou seřazeny další konvoluční vrstvy s postupně se zmenšující velikostí [18].

Vrstvy příznaků se rozdělí do mřížky (viz obrázek 3.7). U každé buňky v této mřížce je pak při detekci predikován relativní posun oproti původnímu ohraničení a vektor o velikosti počtu tříd (s jednou třídou pro pozadí navíc) s hodnotami reprezentující jistotu, že se ve výseči nachází objekt z dané třídy.

V porovnání s modelem Faster R-CNN vyniká SSD především rychlostí. Autoři uvádějí schopnost provádět detekce při 59 snímcích za sekundu (na Nvidia Titan X), což je výrazný nárůst oproti Faster R-CNN. Při testování na datasetu VOC2007 dosáhl SSD přesností mAP 74,3 % u SSD300 a 76,9 % u SSD512.



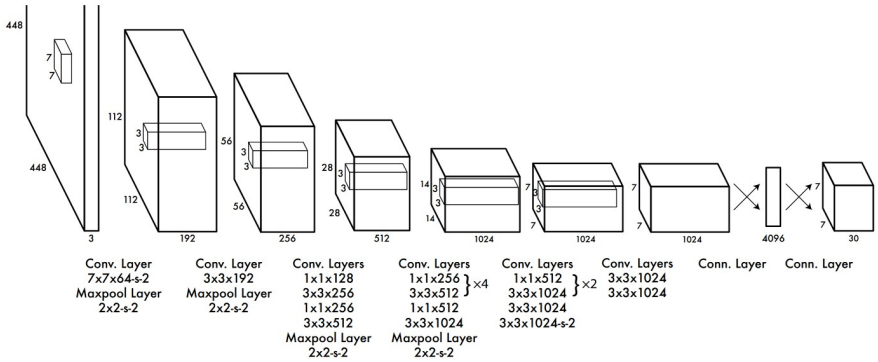
(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Obrázek 3.7: Detekce pomocí SSD. Model pro trénování potřebuje pouze vstupní obraz a rámce ohraničující objekty (a). SSD provádí detekce ve výsečích s různým poměrem stran. Na obrázku (b) je nalezena shoda pro jeden objekt ve dvou různých výsečích. Na obrázku (c) pak shoda s jednou výsečí s většími rozměry [18].

3.3 You Only Look Once

Většina algoritmů přistupuje k problému detekce objektů ve dvou fázích. Vyhledání hraničních oblastí objektů a následné rozpoznání těchto oblastí, často pomocí konvoluční sítě určené pro řešení klasifikačních problémů. Model **You Only Look Once (YOLO)** [23] pomocí jediné neuronové sítě vyhledává kandidátní oblasti a zároveň zajišťuje klasifikaci objektů. Protože je celý proces proveden jednou sítí, je algoritmus YOLO oproti dvoufázovým přístupům mnohem méně výpočetně náročný, při zachování srovnatelné chybovosti.

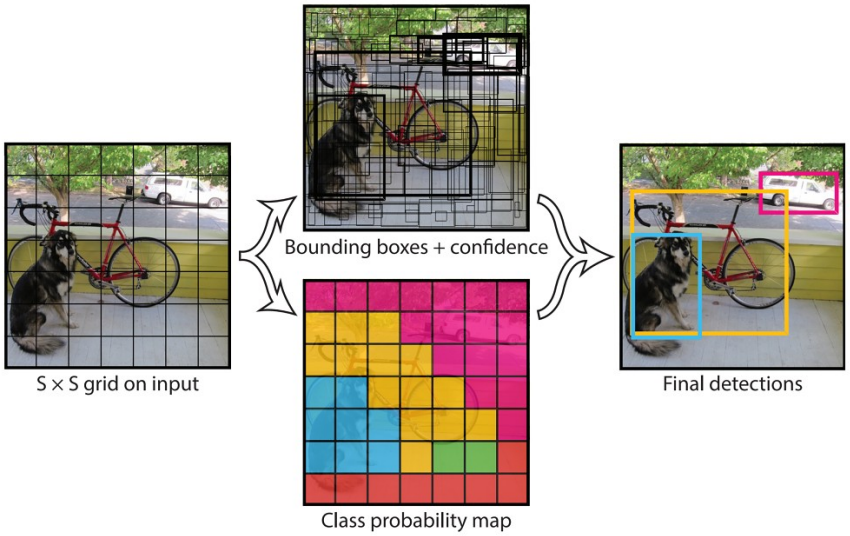
Kromě rychlejší detekce je výhodou modelu YOLO i skutečnost, že při trénování a testování (na rozdíl od metod založených na posuvném okně nebo technik *region-proposal*) vidí celý obraz a je tedy schopen zpracovat kromě samotného vzhledu i kontextuální informace o objektech. Tento přístup také přispívá k robustnosti modelu a algoritmus tak méně chybuje v situacích, kdy pracuje s typem dat, se kterými se dosud nesetkal.



Obrázek 3.8: Architektura YOLO je inspirováná sítí GoogLeNet pro klasifikaci obrazových dat. Síť se skládá z 24 konvolučních vrstev, následovaných dvěma plně propojenými vrstvami [23].

Při detekci algoritmus rozdělí vstupní obraz na mřížku o velikosti $S \times S$. Pokud se střed objektu nachází v dané buňce, je pak tato buňka zodpovědná za detekci pro tento objekt. Každá z buněk poté predikuje určitý počet ohraničených oblastí a skóre, které odpovídá tomu, jak jistý si je model touto vybranou oblastí a zároveň pravděpodobnost, že se objekt v této oblasti nachází. V ideálním případě bude skóre nulové, pokud se v oblasti nenachází žádný objekt. Pokud se v oblasti objekt nachází, bude skóre rovno poměru průniku a sjednocení (*Intersection over Union* - IOU) [23].

Každá ohraničená oblast je definována souřadnicemi x a y, které značí relativní střed oblasti vzhledem k hranicím buňky, dále šířkou a výškou a skórem jistoty detekce, vyjádřeném pomocí IOU. Každá buňka pak generuje C predikcí tříd. Pro každou buňku je generován pouze jeden vektor predikcí a to nezávisle na počtu ohraničených oblastí.



Obrázek 3.9: Model detektoru YOLO rozdělí vstupní obraz na mřížku s $S \times S$ buňkami a pro každou buňku vytvoří B hraničních oblastí, jistotu správnosti ohraničení a vektor predikcí tříd C. Výsledný tenzor je pak vyjádřen vztahem: $S * S * (B * 5 + C)$ [23].

YOLOv2 (původní název – YOLO9000) [24] je vylepšení původního algoritmu. Ten se potýkal s několika problémy, především s vysokým množstvím lokalizačních chyb v porovnání s detektorem Faster-RCNN, který byl v té době přesnější. Autoři se však z důvodu nárůstu potřebného výkonu chtěli vyvarovat použití hlubší sítě, proto pouze původní model upravili několika změnami.

Normalizace dávek trénovacích dat (*batch normalization*) jejich průměrem a odchylkou výrazným způsobem pomáhá ke konvergenci modelu [12]. Z původního modelu YOLO byly odstraněny dropout metody, které zabraňovaly přetrénování, a použitím batch-normalization na všech konvolučních vrstvách bylo dosaženo zlepšení přesnosti modelu.

Dalším zlepšení přineslo použití klasifikátoru na data s větším rozlišením. Většina současných detektorů pracují s rozměry 256×256 pixelů (AlexNet) nebo menší. U původní modelu YOLO byl klasifikátor trénován na datech s rozměry 224×224 pixelů a samotné detekce byly prováděny na dvojnásobném rozlišení. YOLOv2 při trénování v prvních epochách pracuje s rozlišením 448×448 pixelů, díky čemuž je síť robustnější při detekcích na vstupech s vyšším rozlišením.

Oproti původnímu algoritmu byly z modelu také odstraněny plně propojené vrstvy. Namísto nich se na lokalizaci možných oblastí využívá *anchor boxes*, podobně jako u Faster R-CNN. Po skončení učení pak síť pracuje se vstupem o velikosti 416 pixelů, namísto 448. Autoři argumentují, že tímto způsobem mohou umístit jednu buňku na střed obrazu, což je pro algoritmus výhodné. Většina fotografií má objekt umístěný ve středu obrazu a při použití mřížky se sudým počtem buněk by střed obrazu tvořily čtyři sousední buňky. S velikostí 416 pixelů je možné oblast rozdělit po 32 pixelech na mapu příznaků o velikosti 13×13. Protože model využívá pouze konvoluční a pooling vrstvy, je možné v průběhu trénování měnit velikost vstupních dat. Tyto rozměry jsou zvoleny náhodně pro každých 10 dávek dat, musí být ale násobkem 32.

YOLOv3 z roku 2018 opět iterativně vylepšuje předešlé modely. YoloV2 využíval pro predikci tříd softmax, nicméně autoři argumentují, že tento postup není nutný pro přesnou klasifikaci a namísto toho využívají funkci *cross – entrophy*. Tento přístup je výhodný u komplexnějších datasetů, jako je například Open Images Dataset (blíže popsán v kapitole 4.1.2), kde se jednotlivé třídy mohou překrývat (například třída *žena* a *osoba*). Softmax předpokládá, že objekty spadají exkluzivně do jedné třídy, což mnohdy nemusí odpovídat skutečnosti [25].

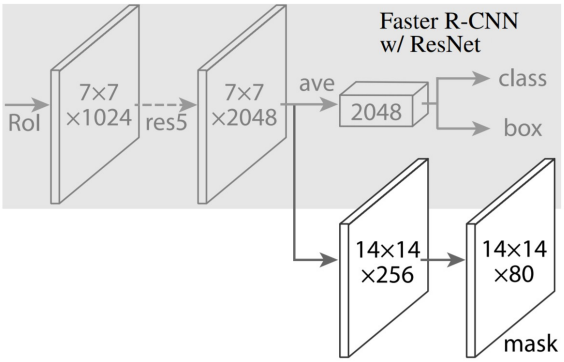
Pro získání příznaků je využita nová architektura inspirovaná sítí ResNet. Tato nová páteřní síť obsahuje 53 konvolučních vrstev (3×3 a 1×1 konvoluce) oproti 19 z YoloV2 a je díky tomu výrazně přesnější. Oproti sítím ResNet dosahuje Darknet53 srovnatelné přesnosti a rychlejšího zpracování.

3.4 Mask R-CNN

Mask R-CNN je model vyvinutý výzkumným týmem Facebooku. Na rozdíl od ostatních detektorů, jejichž výstupem jsou souřadnice ohraničení objektu, Mask R-CNN provádí řazení do jednotlivých kategorií pro každý pixel obrazu. Tento problém se nazývá segmentace objektů (*Object Instance Segmentation*) a je znázorněn na obrázku 3.11.

Metoda Mask R-CNN [6] rozšiřuje model Faster R-CNN přidáním větve pro predikci masky objektů, která pracuje paralelně s již existující větví pro rozeznání hraničních oblastí. Model Mask R-CNN tedy není o mnoho komplexnější než starší detektory a z tohoto důvodu je srovnatelně výpočetně náročný.

Autoři detektoru zjistili, že oblasti v mapě příznaků, které byly vybrány metodou RoI-Pooling, neseděly přesně na oblasti z původního obrazu ze vstupu. Na rozdíl od pouhé lokalizace objektu je při segmentaci objektů v obraze nutná přesnost na pixely, což vedlo k nepřesnostem. Tyto nepřesnosti vznikají při převodu souřadnic z původního obrazu na souřadnice mapy příznaků. V případech, že podíl rozměrů těchto dvou bitmap nebylo celé číslo, docházelo obvykle k zaokrouhlení a tím k následným nepřesnostem. Autoři tento problém řeší použitím metody zvané *RoIAlign*, která využívá bilineární interpolaci pro přesnější přiřazení pixelů z původního obrazu k pixelům mapy příznaků, díky čemuž lze dosáhnout přesnějšího zarovnání. Po vygenerování masek dojde ke kombinaci s predikcemi hraničních oblastí, díky čemuž Mask-RCNN dosahuje velmi přesných lokalizací s predikcemi jednotlivých tříd.



Obrázek 3.10: Model Mask R-CNN přidává plně propojenou síť (v obrázku bílou barvou) na vrstvy příznaků z Faster R-CNN. Výstupem je maska segmentací, která je generována paralelně s rozpoznáváním hraničních oblastí [6].

Obrázek 3.11: Ukázka segmentace objektů pomocí Mask R-CNN. K hraničním oblastem a predikcím tříd generovaným pomocí Faster R-CNN přidává i segmentaci objektů po jednotlivých pixelech. Tyto oblasti jednoho objektu jsou vyznačeny v odstínech stejné barvy [6].

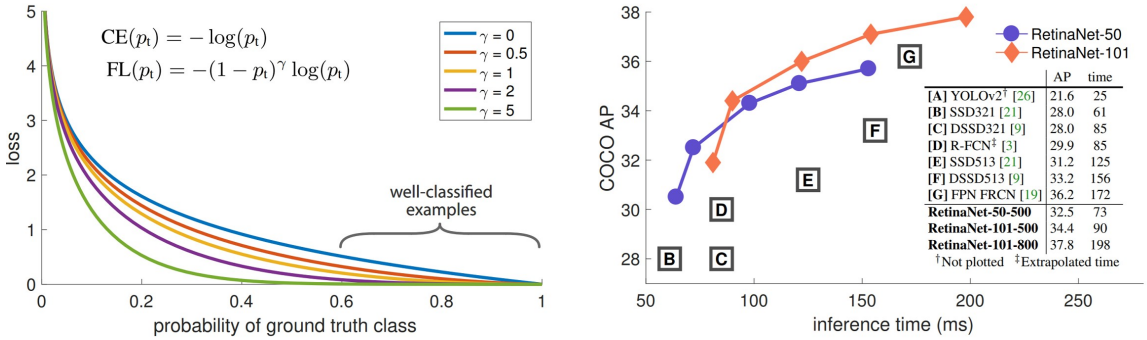
3.5 RetinaNet

Model RetinaNet je jednostupňový detektor objektů v obraze, který díky nově navrhnuté chybové funkci *focal loss* dosahuje přesnosti srovnatelné s dvoustupňovými detektory [17]. Autoři identifikovali jako hlavní problém jednostupňových modelů (SSD, DPM) nevyrovnanost jednotlivých tříd (*class imbalance*). Tyto detektory vyhodnocují mezi 10^4 a 10^5 možných oblastí na každý vstupní obraz. Z těchto případů ale pouze zlomek skutečně obsahuje objekt, někdy je poměr výřezů s objekty a s pozadím až 1:1000. Takto vysoký rozdíl způsobuje problémy, které se autoři snaží řešit pomocí funkce *focal loss*.

Trénování detektoru je při vysoké nevyrovnanosti tříd neefektivní, protože většina oblastí jsou lehce rozpoznatelné negativní detekce, neboť se jedná o výřezy obsahující pozadí. Takto vysoký počet lehkých detekcí má za následek velmi pomalé zlepšování detekcí nebo může model degenerovat. Běžným řešením tohoto problému je tzv. *hard negative mining*, což je algoritmus, díky kterému se model učí více z těžkých vstupů, u kterých chybně

provedl detekci. Autoři modelu RetinaNet toto řeší chybovou funkcí *focal loss*, která přiřazuje větší váhu náročným detekcím a nižší těm jednodušším. Srovnání funkce *focal loss* a *cross-entrophy loss* je v grafu 3.12.

Pro demonstraci využití funkce *focal loss* vytvořili autoři sít RetinaNet. Ta využívá dopřednou sít typu ResNet, za kterou je napojena Feature Pyramid Network (FPN). Následně se síť dělí do dvou větví, jedna pro detekování oblastí z objektu a druhá pro určení třídy objektu.



Obrázek 3.12: V levém grafu je znázorněno porovnání funkce *cross entrophy loss* (CE) a funkce *focal loss* (FL). CE (v grafu vyznačena modrou barvou) přiřazuje relativně vysokou hodnotu i v jednoduchých případech. FL toto redukuje v závislosti na parametru γ. V pravém grafu je pak srovnání rychlosti (ms) a přesnosti (AP) detektoru RetinaNet s ostatními modely [17].

3.6 Shrnutí a výběr detekční sítě

Při výběru sítí pro trénování bylo bráno v potaz několik kritérií. Hlavní požadavky byly přesnost detekcí a rychlost modelu. Z těchto důvodů byly z výběru vyřazeny modely z rodiny R-CNN. Ačkoliv se jedná o ověřené a často používané detektory, jejich přesnost a rychlost byla již překonána novějšímy modely. Z podobných důvodů nebyl trénován model SSD.

Model Mask RCNN je vysoce přesný detektor, nicméně pro účely této práce není nutná segmentace obrazu po pixelech. Trénování tohoto modelu by také vyžadovalo použití datasetu obsahujícího segmentované masky objektů, namísto hraničních boxů, což výrazně limituje výběr datasetu. Z těchto důvodů nebyl tento detektor zvolen pro trénování.

Ze zbývajících modelů byl vybrán detektor RetinaNet. Ten nabízí v současné době přesnost překonávající ostatní algoritmy, při zachování rozumné rychlosti a je proto vhodným modelem pro účely této práce. Především z důvodů vysoké rychlosti zpracování snímku byl pro trénování také zvolen model Tiny-Yolo. Ten, ačkoliv v přesnosti zaostává, je jako jeden z mála modelů schopen relativně plynulého běhu na CPU. Jako třetí detektor byl vybrán nejnovější model YoloV3, který byl v době vzniku této práce spíše v experimentální fázi vývoje. Jeho přesnost a rychlost je podobná detektoru RetinaNet, ale oproti němu umožňuje provádět detekce na vstupu variabilním rozlišením, což při jeho využití výhodné a může výrazně zrychlit tento jinak spíše pomalý detektor.

Kapitola 4

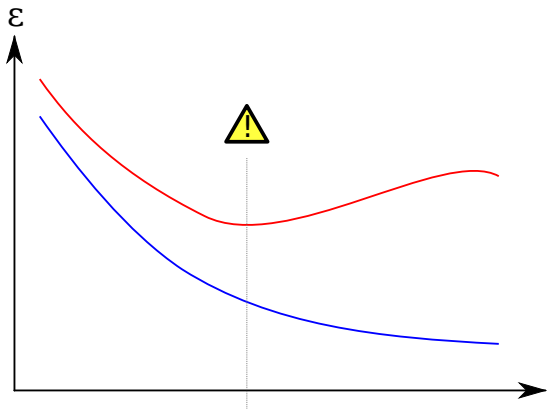
Implementace a trénování detektorů

Trénování detektorů probíhalo na počítačích v národním superpočítačovém centru IT4Innovations, které je součástí Vysoké školy báňské – Technické univerzity Ostrava. Tato organizace poskytuje studentům a zaměstnancům vysokých škol možnost zdarma využít výpočetní prostředky pro výzkumné účely. Pro tuto práci šlo hlavně o využití grafických karet, které mají podporu CUDA a knihovny cuDNN, které výrazným způsobem akcelerují výpočty nutné pro trénování neuronových sítí. Konkrétně byly využity uzly s GPU akcelerátorem NVIDIA Kepler K20 (2496 jader CUDA) s 6 GB paměti GDDR5, dvěma procesory Intel Sandy Bridge E5-2470, 2,3 GHz a 96 GB operační paměti. Pro přípravu projektu a správu dat je k dispozici lokální 500 GB disk.

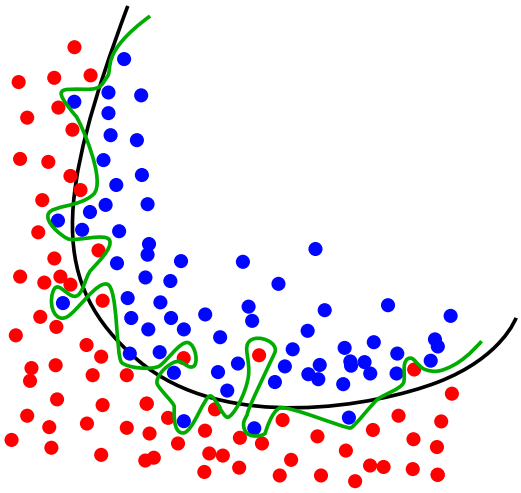
Pro trénování většiny sítí byla využita knihovna Tensorflow a nad ní běžící framework Keras ve verzi 2.0.5. Tento framework umožňuje vysokoúrovňovou práci s neuronovými sítěmi, rychlé prototypování a podporou GPU akcelerace. Většina výzkumu v oblasti neuronových sítí využívá pro tyto účely právě kombinaci knihoven Tensorflow a Keras. Framework Keras byl vytvořen v programovacím jazyce Python a stejně tak i samotný kód detektorů je psaný v tomto jazyce. Výjimku tvořil detektor YoloV3, který byl trénován ve frameworku Darknet. Ke zpracování obrazu, vizualizaci výsledků a snímání obrazu z webkamery byla použita knihovna OpenCV. Pro práci s daty a alokaci výpočetních prostředků na clusteru Anselm byl využit skriptovací jazyk Bash.

Při trénování modelů je běžné, že se model po určitém počtu iterací dostane do stavu, kdy dokáže velmi přesně predikovat výsledky na datech určených pro trénování (a má tedy velmi malou chybovou funkci), ale při použití nových vstupních dat je méně přesný, než by tomu bylo při použití menšího počtu iterací při trénování. Tento stav se označuje jako přetrénování modelu (*overfitting* nebo také *overtraining* [34]) a je znázorněn na grafu 4.1. Při skutečném použití je sada dat pro trénování omezená a model se tak po určité době začne zaměřovat na prvky obrazu, které jsou specifické pro danou sadu dat a detektor tak ztrácí na obecnosti (*generalization*). V extrémním případě si pak model přesně pamatuje všechna data a je tak schopen v rámci trénování podávat bezchybné predikce. Takovéto chování přetrénovaného modelu je zachyceno v grafu 4.2. Přetrénování modelu nastává v momentě, kdy se chybová funkce blíží nule, ale přesnost detektoru na validačních datech se snižuje. Je tedy vhodné trénování modelu v tomto bodě zastavit, popřípadě vybrat snapshot, který byl nejblíže tomuto bodu. Validační data nejsou použita pro samotný trénink, ale slouží pouze

k ověření úspěšnosti modelu během trénování. Na základě těchto informací pak mohou být upraveny hyperparametry pro dosažení přesnějších výsledků.



Obrázek 4.1: Příklad vývoje chybové funkce na trénovací množině dat (modrou barvou) a validační množině dat (červenou barvou) v závislosti na počtu iterací. K přetrénování dochází v momentě, kdy hodnota validační chyby přestává klesat. V tento moment je vhodné trénování přerušit[1].



Obrázek 4.2: Zelená křivka značí přetrénovaný model, zatímco černá křivka reprezentuje obecný, správně natrénovaný model. Zatímco zelená křivka se lépe přizpůsobuje trénovací množině dat, ztrácí schopnost správně klasifikovat neznámá data[2].

4.1 Datasety

Pro úspěšné natrénování a evaluaci detektorů použitých v této práci bylo nutné obstarat dataset obsahující fotografie lidských hlav a souřadnice vymezující polohu hlavy v obraze (anotovaný dataset). Zatímco pro rozpoznání obličeje nebo lidské postavy existuje řada kvalitních zdrojů dat, pro detekci hlavy v různém natočení ke kameře je výběr datasetu limitovaný. Při výběru byl kladen důraz na velikost datasetu, přesnost anotací a různorodost dat.

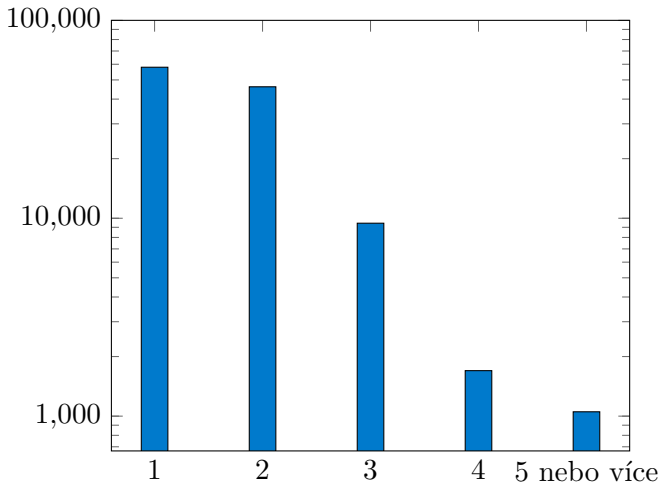
4.1.1 HollywoodHeads

Dataset HollywoodHeads vznikl extrakcí snímků z hollywoodských filmů [35]. Dataset obsahuje celkem 224 740 snímků z 21 filmů na kterých je zachyceno celkem 369 846 lidských hlav. Z celkového počtu 21 filmů bylo 15 použito na trénování, tři na validaci a zbylé tři na testování modelů. Výběr filmů pokrývají různé filmové žánry a různá období vzniku, dva z filmů byly natočeny černobíle. Rozlišení snímků se lišilo podle filmu, obvykle se jednalo o hodnoty pod 600×350 pixelů. Filmy se v rámci rozdělení na část pro trénování, validaci a testování nepřekrývají, takže nehrozí ovlivnění výsledků z důvodu podobnosti dat.

[1] https://en.wikipedia.org/wiki/File:Overfitting_svg.svg
[2] https://en.wikipedia.org/wiki/File:Overfitting.svg

Autoři datasetu uvádějí, že anotace byly tvořeny manuálně pro několik klíčových snímků a pozice hlav na snímcích mezi nimi byly interpolovány. Tyto anotace jsou ke každému snímku uloženy ve zvláštním souboru ve formátu xml, který obsahuje kromě informací o datasetu také rozměry obrázku, jeho barevnou hloubku a pro každou zachycenou hlavu také souřadnice. Celkový formát anotací odpovídá datasetům Pascal Visual Object Classes (VOC) [2], což výrazně usnadňuje trénování modelů, protože ty jsou ve většině případů schopny tento formát zpracovat bez úprav. Pro trénování některých modelů bylo nutné vyřadit fotografie, které buď neobsahovaly žádnou hlavu, nebo v jejich anotacích byly údaje, se kterými si model neuměl pracovat. Jednalo se například o záporné hodnoty souřadnic nebo chybějící tagy v anotačních souborech.

Obrázek 4.3: Ukázka datasetu HollywoodHeads.



Obrázek 4.4: Rozložení počtu anotovaných objektů na snímek v datasetu HollywoodHeads. Jelikož jsou zdrojem dat filmy, valná většina snímků obsahuje jednu nebo dvě lidské hlavy.

4.1.2 Google Open Images

Dataset Google Open Images se skládá z asi 9 milionů odkazů na anotované obrázky pro detekci objektů z více než tisíce tříd, mezi nimiž se nachází i třída pro lidskou hlavu. Správným protříděním dat a anotací je tedy možné získat podmnožinu datasetu vhodnou pro trénování detektorů použitých v této práci. Využity však mohly být fotografie pouze určené pro validaci a testování, neboť v datasetu pro trénování se nacházely případy, kdy nebyly všechny lidské hlavy správně zaznačeny a modely by při trénování na těchto snímcích zřejmě nekonvergovaly.

Jelikož se dataset Google Open Images skládá pouze z odkazů, mohlo nastat, že některé vedly na již neexistující soubor. Fotografie z datasetu jsou uloženy na serveru Flickr.com, který v těchto případech zobrazí prázdný bílý obrázek. Tyto případy bylo nutné z datasetu vyřadit. Jednalo se o 392 obrázků, tedy asi 3 % datasetu. Ukázka protříděného datasetu je zobrazena na obrázku 4.5.

Obrázky v datasetu mají různé rozlišení, obecně se jedná spíše o fotografie s většími rozměry. Taktéž poměru stran se liší, některé z obrázků jsou černobílé. V datasetu se nachází i fotografie, na kterých se vyskytují větší skupinky lidí, což může vést k vyšší robustnosti výsledného detektoru. V několika málo případech jsou jako lidská hlava označeny i hlavy v malbách nebo hlavy u soch. Je těžké odhadnout, zda anotace tohoto typu ovlivní výslednou úspěšnost detekcí. Některé takovéto obtížnější případy byly z datasetu manuálně vyřazeny (viz obrázek 4.6). Vysoká různorodost fotografií je však vhodná především pro subjektivní porovnání detekcí v různých typech scény (viz kapitola 5.3).

Autoři datasetu uvádí, že anotace byly nejprve získány strojově a následně proběhla jejich manuální kontrola a korekce. Anotace se nachází v csv souboru, kde každý řádek značí pozici jednoho rámečku. Kromě pozice se zde nachází i informace o třídě (pro účely této práce pouze třída */m/04hgtk* značící lidskou hlavu), zdali je objekt zakrytý jiným objektem nebo jestli se celý nachází v obrázku.

Obrázek 4.5: Ukázka datasetu Google Open Images vyfiltrovaného na obrázky, ve kterých byla anotována hlava.

Obrázek 4.6: Ukázka případů, které byly z datasetu Google Open Images vyřazeny. Ražba na minci, malba na antické váze a portrét kreslený tuží sice obsahují lidskou hlavu, nicméně jejich zobrazení je pro trénování detektoru příliš abstraktní.

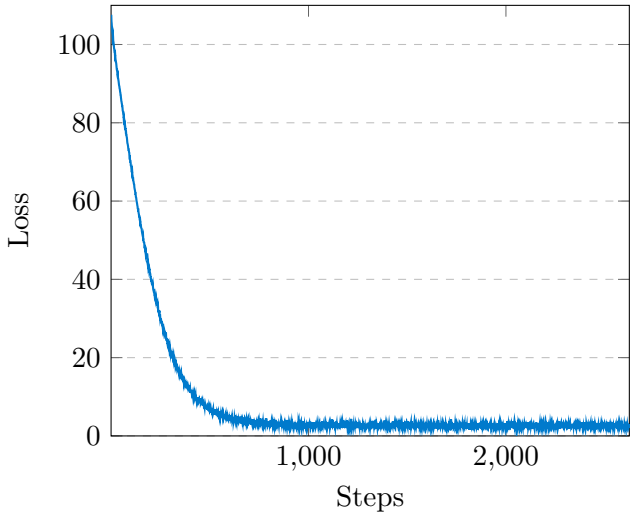
4.2 Trénování detektoru TinyYolo

Původní modely z rodiny YOLO byly vytvořeny ve frameworku Darknet. Tato open source knihovna, napsaná v jazyku C, podporuje akceleraci výpočtů pomocí GPU. Nicméně instalace tohoto frameworku na vzdálený server se ukázala jako problematická a oproti jiným frameworkům je Darknet pomalý. Z těchto důvodů byla pro účely této práce využita verze detektoru pod názvem Darkflow. Jedná se Darknet přeložený do knihovny Tensorflow. Pro běh jsou kromě Python3 vyžadovány i knihovny numpy, OpenCV3 a cython. Cluster Anselm má všechny tyto knihovny předinstalované, stačí pouze načíst příslušné moduly.

Sítě typu YOLO jsou jednostupňové detektory, populární jsou především pro svou rychlost při zachování přesnosti srovnatelné s dvoustupňovými detektory. Z existujících sítí typu YOLO připadají v úvahu sítě YOLOv2 a TinyYolo. Především z důvodu rychlosti byl zvolen model TinyYolo. Jedná se o jediný detekční model, který dokáže detekovat objekty v reálném čase za běhu na CPU při zachování dobré úspěšnosti detekcí.

Pro zrychlení trénování sítě byly původní hodnoty vah inicializovány z již vytrénované sítě. Jedinou změnou byly snížení počtu tříd objektů na jednu. K novému počtu tříd je nutné změnit i počet filtrů v poslední konvoluční vrstvě. Jejich počet je nutné v konfiguračním souboru přepsat podle vztahu $5*(3+N)$, kde N značí počet tříd. Pro samotné trénování byl zvolen dataset HollywoodHeads, jelikož obsahoval více snímků a Darkflow umí s datasety ve formátu Pascal VOC pracovat přímo a není tak zapotřebí upravovat anotační soubory.

Učící krok byl z počátku ponechán na původní hodnotě e^{-5}. Na kroku 10 376 byl pak z důvodu stagnující hodnoty loss snížen na 0.0001. Po této úpravě však chyba modelu dále spíše stagnovala. Celkem bylo provedeno 38 000 iterací, checkpoint s vahami byl ukládán každých 2000 iterací.



Obrázek 4.7: Vývoj chybové funkce v prvotních krocích učení modelu Tiny-Yolo.

4.3 Trénování detektoru RetinaNet

Pro trénování detektoru RetinaNet byla zvolen model napsaný ve frameworku Keras nad knihovnou Tensorflow. Tyto knihovny byly již předinstalované na server Anselm a to i s podporou akcelerace výpočtů pomocí GPU. Pro trénování tak opět stačilo pouze načíst příslušné moduly.

Při učení se celková chybová funkce počítá jako součet dvou složek, klasifikační chyby (*classification loss*) a regresní chyby (*regression loss*). Klasifikační složka chyby je definována jako funkce focal loss [17]. Definice této funkce vychází z funkce Cross Entrophy Loss pro binární klasifikaci

$$CE(p, y) = \begin{cases} -\log(p) & \text{pro } \ y = 0 \\ -\log(1-p) & \textit{jinak} \end{cases} \tag{4.1}$$

kde $y \in \pm 1$ značí třídu a $p \in [0, 1]$ je jistota predikovaná modelem pro třídu s označením $y = 1$. Pro zjednodušení notace je definováno p_t:

$$p_t = \begin{cases} p & \text{když } y = 1 \\ 1-p & \textit{jinak.} \end{cases} \tag{4.2}$$

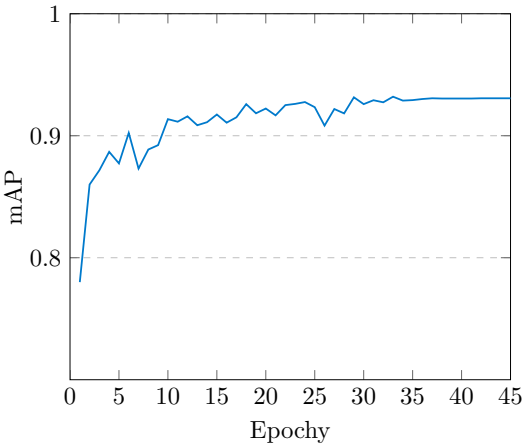
a funkci $CE(p, y)$ lze tedy definovat jako $CE(p_t) = -\log(p_t)$. Samotná funkce focal loss je tedy pak definována jako

$$FL(p_t) = -\alpha(1-p_t)^{\gamma} * \log(p_t). \tag{4.3}$$

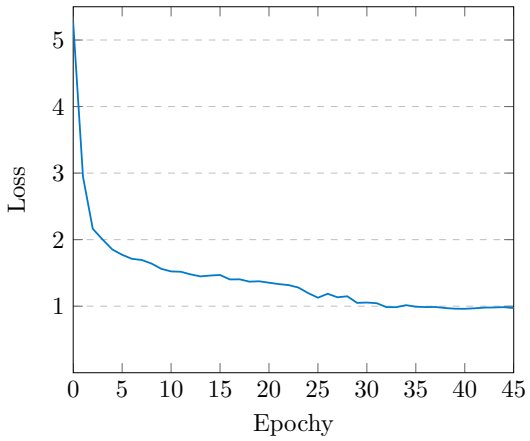
V rámci implementace této funkce byly zvoleny hodnoty $\alpha = 0.25$ a $\gamma = 2$. Vývoj chybové funkce v průběhu učení sítě je zachycen v grafu 4.9 a vývoj úspěšnosti nad validačními daty v grafu 4.8.

Pro samotné trénování byla zvolena hodnota 2000 iterací na epochu. Po dokončení každé epochy byla provedena evaluace úspěšnosti detektoru. Trénování běželo na GPU Tesla K20 celkem 24 hodin, během kterých proběhlo 46 epoch. V pozdějších fázích byly již úbytky

chybové funkce minimální, stejně tak úspěšnost detektoru začala stagnovat. Pro trénování byl zvolen dataset HollywoodHeads a u každého obrázku byla 50 % šance, že bude před vstupem do sítě zrcadlově převrácen.



Obrázek 4.8: Vývoj úspěšnosti detektoru RetinaNet nad validačními daty v závislosti na počtu iterací.



Obrázek 4.9: Vývoj chyby detektoru RetinaNet v průběhu trénování.

4.4 Trénování detektoru YoloV3

Model YoloV3 byl v době psaní této práce implementován pouze ve frameworku Darknet, což je *open-source* framework na tvorbu neuronových sítí. Především je využíván k výzkumným účelům a na testování modelů z rodiny Yolo. Pro trénování sítě bylo nutné převést anotace z formátu Pascal VOC do formátu, který je framework schopný pracovat. Pro každý snímek z trénovacího datasetu byl vytvořen textový soubor, kde pro každý objekt v obraze byly zapsány údaje o jeho pozici a třídě objektu ve tvaru: `class x y šířka výška`.

Pro trénování sítě je také nutné upravit nastavení některých parametrů v konfiguračním souboru. Především se jedná o informace o počtech tříd a počty filtrů v některých konvolučních vrstvách. Počet filtrů se odvíjí od vztahu $(N + 5) * 3$, kde N značí počet tříd v datasetu. Velikost dávky pro trénování byla nastavena na hodnotu 64 a velikost učícího kroku na 0.001. Vstupní rozlišení do sítě bylo ponecháno na hodnotách 416×416, nicméně tuto velikost lze později libovolně upravovat na násobky 32 bez nutnosti síť na nové rozlišení přetrénovat. Vyšší rozlišení přispívá k přesnějším detekcím za cenu pomalejšího výpočtu.

Při trénování je po každých 1000 zpracovaných dávkách uložen aktuální stav sítě. Z těchto souborů lze pak provést evaluaci modelu. Celkem bylo při trénování provedeno 10 000 iterací, nejpřesnější výsledky na validačních datech byly naměřeny po 7000 iteracích.

4.5 Implementace aplikace

Jednou z částí teto práce je implementace aplikace, která bude demonstrovat detekce hlav na vstupních obrazových datech. Při výběru vhodného modelu byl kladen důraz především na přesnost detekcí, v menší míře poté na rychlost zpracování snímku. Vzhledem k požadavku na multiplatformnost aplikace byla také brána v potaz složitost přenosu aplikace na různé operační systémy a komplikace, které by mohly vzniknout uživateli při instalaci potřebných knihoven a modulů.

Při porovnání současných *state-of-the-art* detektorů měl detektor RetinaNet při detekci lidské hlavy nejmenší počet chybných detekcí (podrobněji v kapitole 6). Model YoloV3 dosahoval nižší přesnosti při podobných požadavcích na výkon a model Tiny-Yolo měl ve srovnání s ostatními detektory vysokou chybovost. Z těchto důvodů byl pro tuto aplikaci zvolen právě detektor RetinaNet.

Při volbě páteřní sítě detektoru byly zvažovány sítě typu ResNet a MobileNet. Síť MobileNet [10] je klasifikační síť určená pro mobilní a embedded zařízení, kdy je zapotřebí zrychlit a zjednodušit výpočty při detekci. Při testování detektoru RetinaNet se sítí MobileNet však byla úspora času při výpočtu detekcí na jeden snímek zanedbatelná. Z tohoto důvodu byla jako páteřní síť ponechána ResNet-50, která v klasifikačních úlohách dosahuje nižší chybovosti.

Samotná aplikace je naprogramována v jazyce Python3. Pro správné načtení knihoven je nutné použít 64 bitovou verzi, protože některé knihovny nejsou podporovány v 32 bitovém režimu. Pro vývoj aplikace bylo využito open-source vývojové prostředí Visual Studio Code. Pro práci s neuronovou sítí byla použita knihovna Tensorflow (ve verzi 1.7) a nad ní běžící framework Keras (verze 2.0.5). Pro načítání a zpracování obrazových dat byla použita knihovna OpenCV (ve verzi 3.4.0). Všechny tyto nástroje jsou platformě nezávislé a aplikaci tak lze spustit na většině operačních systémů, potřebné knihovny lze nainstalovat pomocí správce balíků Pip[3]. Pro účely této práce byla aplikace testována na Windows 10 (64 bit, OS Build 16299.371) a Ubuntu 16.04.

Po spuštění aplikace je nejprve načten soubor s uloženým modelem sítě a vahami. Výchozí cesta k souboru je nastavena k natrénovanému modelu s páteřní sítí ResNet50, ale uživatel může využít i libovolný vlastní soubor ve formátu h5. Aplikace následně provede načtení vstupních snímků. Jako zdroj těchto snímků může být přímo obrázek nebo složka s obrázky, ze které budou načteny všechny souboru ve formátu jpg a png. Aplikace také umožňuje zpracovat snímky z videa nebo z připojené webkamery. V případě, že je aplikace spuštěna pouze na procesoru, je při zpracovávání snímků z videa doporučeno pomocí argumentu specifikovat, kolik snímků lze při detekci vynechat. Pro rozpoznání nebo trakování objektů ve videu není obvykle nutné zpracovávat každý snímek a jejich vynecháváním tak lze výrazně urychlit zpracování videa.

Jednotlivé snímky jsou následně převedeny do BGR barevné reprezentace a je upravena jejich velikost tak, aby šířka nepřesahovala 1024 a výška 600 pixelů. Obrázky nejsou de-

[3] https://github.com/pypa

formovány a výsledné rozlišení se tak liší v závislosti na poměru stran původního snímku. Na upraveném obrázku je pak provedena detekce a jejím výstupem je pole predikcí, kde každá položka obsahuje souřadnice detekované oblasti, predikovanou třídu a ohodnocení pravděpodobnosti výskytu objektu. Detekce, kde je tato pravděpodobnost nižší než hodnota *threshold*, jsou ignorovány. Proměnná *threshold* je ve výchozím nastavení 0,5, uživatel může tuto hodnotu specifikovat použitím příslušného argumentu.

Hraniční oblasti, u kterých je pravděpodobnost výskytu objektu dostatečně vysoká, jsou pomocí OpenCV zakresleny do původního obrázku a ten je zobrazen uživateli. Pokud je zdrojem video nebo webkamera, dojde k okamžitému zpracování následujícího obrázku. V opačném případě jsou nové detekce zjištěny po interakci uživatele. Výsledné detekce lze uložit na disk ve formě obrázku nebo videa. Celý algoritmus zpracování snímků aplikací je zobrazen níže.

Algoritmus 1: Pseudokód aplikace detektoru

Vstup:
threshold: mezní hodnota úspěšné detekce
model: uložený soubor modelem a vahami sítě
images: zdroj obrazových dat

```
Zpracování a validace parametrů
Načtení modelu
Načtení snímků z disku
for image in images[] do
    Převod obrázku do BGR
    Zmenšení obrázku na 1024 × 600
    Detekce objektů v obrázku
    for detekce v detekcích do
        if skóre detekce < threshold then
            continue
        end if
        Zakreslení boxu
        Zakreslení skóre
    end for
    Zobrazení obrázek
end for
Uvolnění prostředků a ukončení
```

4.5.1 Dotrénování detektoru na vlastních anotacích

Pro úspěšné trénování detektorů je klíčové množství dat a správnost anotací, ze kterých se síť naučí rozpoznávat hledané objekty. V praxi často nastane situace, že není z časových nebo finančních důvodů možné tvořit nový dataset, který by obsahoval data podobná těm, které bude detektor následně vyhledávat, a je proto nutné použít pro učení již nějaký exitující dataset. Je však pravděpodobné, že tato volně dostupná data budou do jisté míry

odlišná (rozlišení, vzdálenost objektu od kamery, počet objektů na jeden snímek) od těch, se kterými bude pracovat natrénovaný detektor. Tyto odlišnosti v trénovacích a testovacích datech mohou způsobit snížení úspěšnosti modelu oproti případům, kdy by na trénování byla použita podobná obrazová data.

Pro vyřešení tohoto problému je v této práci zkoumán postup učení sítě, kdy je detektor nejprve natrénován na obecných datech z volně dostupného datasetu a následně probíhá doučení tohoto detektoru na neoznačených snímcích, kde anotace vytváří sám obecně natrénovaný detektor. V praxi by tento způsob učení byl použitelný například u bezpečnostních kamer, kdy by se obecně natrénovaný detektor dotrénoval na snímcích z jednoho místa, které bude kamera snímat. Lze očekávat, že model tímto způsobem trénování ztratí na robustnosti, ale zároveň učení na podobných datech povede ke zlepšení detekcí v místech, kde bude detektor skutečně nasazen.

Pro tento způsob trénování byla vytvořena aplikace, které pomocí natrénovaného detektoru převádí video na anotovaný dataset, na kterém je možné jednotlivé modely dotrénovat. Z důvodu nízkého počtu falešně pozitivních detekcí, které by způsobily dotrénování na špatných anotacích, byl opět zvolen detektor RetinaNet.

Program nejprve video rozdělí na jednotlivé snímky. Ty jsou následně předzpracovány, je upravena jejich velikost tak, aby šířka nepřesahovala 1024 a výška 600 pixelů a snímek je převeden do barevné reprezentace BGR. Na každém snímku je pak pomocí RetinaNet provedena detekce objektů. Výsledné detekce, které byly nalezeny s jistotou vyšší než je mezní hodnota *threshold*, jsou uloženy do souboru ve formátu xml se strukturou odpovídající datasetům z Pascal Visual Object Classes Challenge (znázorněno na obrázku 4.10). U každého boxu jsou uloženy souřadnice a informace o třídě, do které objekt spadá. Spolu s tím se ukládají také informace o rozlišení obrázku a barevné hloubce. Samotné snímky z videa jsou ukládány do adresáře `JPEGImages`. V případě, že na snímku nebyly nalezeny žádné platné detekce, je snímek z datasetu vyřazen. Tím se sníží počet případů, kde se v datasetu vyskytují chybně neoznačené objekty.

S tímto programem byl z videa z bezpečnostní kamery[4] vytvořen dataset, kde hledaným objektem byla hlava. Celkový počet snímků v části na trénování byl 1044. Pro vytvoření testovací sady byla použita jiná část záběru ze stejné bezpečnostní kamery. Na těchto snímcích byly manuálně anotovány hlavy pomocí volně dostupného open-source programu labelImg[5], který je schopný vytvářet soubory ve formátu xml se strukturou anotací typu Pascal Visual Object Classes Challenge. Do testovacího části datasetu nebyly zařazeny snímky, kde se nenacházel žádný hledaný objekt. Celkem tak bylo manuálně anotováno 124 snímků, na kterých lze otestovat, zda se přesnost modelu po dotrénování na vlastních anotacích zlepšuje.

[4] Záběry jsou volně dostupné pod licencí Creative Commons:
https://www.youtube.com/watch?v=HdfbkLzH3S4

[5] https://github.com/tzutalin/labelImg

```
<annotation>
    <folder>      newvoc  </folder>
    <file>    frame0.jpg    </file>
    <source />
    <size>
        <width>       1024   </width>
        <height>       576  </height>
        <depth>          3   </depth>
    <object>
            <name>     head    </name>
            <bndbox>
                <xmin>    338    </xmin>
                <ymin>     18    </ymin>
                <xmax>    386    </xmax>
                <ymax>     68    </ymax>
```

Obrázek 4.10: Ukázka strojově generovaného anotačního xml souboru. O obrázku jsou zaznamenány údaje o jeho poloze v adresářové struktuře, jeho rozlišení a barevné hloubce. Každý nalezený objekt je pak zaznamenán pomocí čtyř hodnot reprezentující souřadnice (s počátkem souřadnic v levém horním rohu).

Pro ověření této metody trénování detektorů byl obecně natrénovaný model (RetinaNet natrénovaný na datasetu HollywoodHeads, viz kapitola 4.3. Získaný dataset lze však použít i pro jiný model.) dotrénován na tomto uměle vytvořeném datasetu ze statické bezpečnostní kamery. Trénování se skládalo z desíti epoch, kde délka epochy byla rovna počtu snímků s úspěšnými detekcemi. Model tedy při trénování během každé epochy prošel celý dataset. Po skončení epochy byl vždy uložen snapshot, který obsahoval aktuální stav sítě. Při průběhu trénování neprobíhala evaluace detektoru, namísto toho byla data z každého snapshotu otestována po skončení trénování. Graf s vývojem úspěšnosti je zobrazen v kapitole 5.2.

Kapitola 5

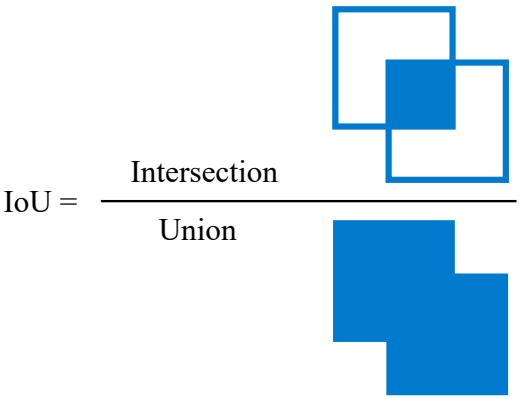
Výsledky experimentů

Pro vyhodnocení úspěšnosti modelů byla použita metoda *average precision*, která je používána na soutěžích Pascal VOC Challenge. Hodnota *average precision* je vypočítána z *precision/recall* křivky [2], kde hodnoty *precision* a *recall* jsou vyjádřeny jako

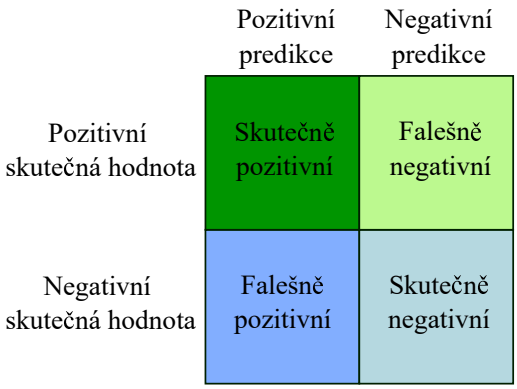
$$Precision = \frac{true\ positives}{true\ positives + false\ positives}, \tag{5.1}$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives}. \tag{5.2}$$

Jako skutečně pozitivní detekce jsou označeny ty detekce, které mají podíl průniku a spojení oblastí vymezených predikovanými a skutečnými hraničními oblastmi (tzv. *Intersection over Union* - IoU) větší než 0,5. Falešně pozitivní detekce jsou případy, kdy detektor označí za nalezený objekt jinou oblast, než kde je objekt anotován. Případy, kdy není anotovaný objekt nalezen, jsou značeny jako falešně negativní.



Obrázek 5.1: Ukázka výpočtu hodnoty IoU. Za správnou detekci jsou považovány ty případy, kdy je podíl obsahu průniku a obsahu sjednocení větší než 0,5.



Obrázek 5.2: Matice predikcí. Pro vyhodnocení úspěšnosti detektoru jsou brány v potaz skutečně pozitivní, falešně pozitivní a falešně negativní případy.

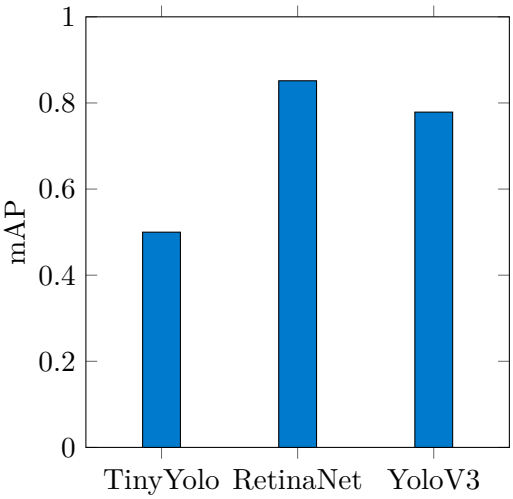
5.1 Úspěšnost a rychlost natrénovaných detektorů

Vyhodnocení úspěšnosti detektorů bylo měřeno na testovací části datasetu HollywoodHeads 4.1.1. Ta se skládá ze snímků ze tří filmů. Jelikož detektory byly trénovány na podobných datech, jako jsou testovací (podobné rozlišení, kompozice a pozice hlav typická pro filmy), lze na tomto datasetu očekávat nejvyšší úspěšnost detekcí.

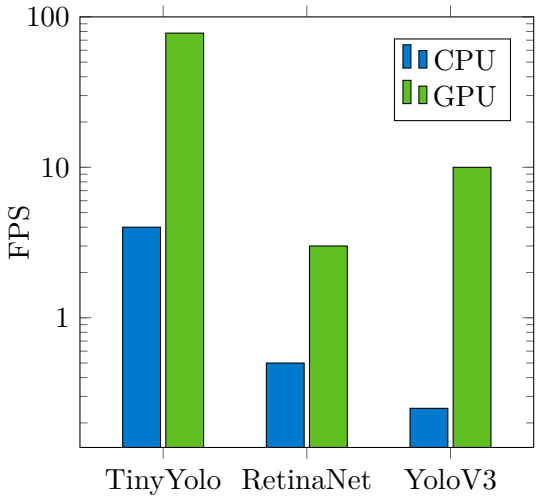
U modelu RetinaNet byla naměřena hodnota 85,15 %, pro jejíž výpočet byl použit snapshot po 19 epochách (dosáhl nejnižší chybovosti na validačních datech, viz obrázek 4.1). Měření rychlost modelu probíhalo na Windows 10 s knihovnou Tensorflow ve verzi 1.6. Rychlost detekcí se pohybovala okolo dvou sekund na vyhodnocení jednoho snímku na procesoru Intel Core i5-4590 taktovaném na 3,3 GHz. Při akceleraci výpočtů pomocí grafické karty Nvidia Tesla K20 se rychlost detektoru zvýšila na 3 snímky za sekundu.

Detektor TinyYolo dosáhl na stejném datasetu úspěšnosti mAP 49,99 % a to po 34 000 iteracích. Ačkoliv se jedná o nižší přesnost, než které dosáhl model RetinaNet, na procesoru Intel Core i5-4590 dokázal model zpracovat 4 snímky za sekundu. Při akceleraci pomocí GPU byla rychlost detektoru 78 snímků za sekundu.

U modelu YoloV3 byla na datasetu HollywoodHeads naměřena 77,87 % mAP. Při běhu na procesoru dosahoval model rychlosti zpracování jednoho snímku za 4 sekundy. Při akceleraci výpočtů na GPU se rychlost pohybovala okolo 10 snímků za sekundu. Rychlost byla měřena s velikostí vstupu 416×416 pixelů. U detektoru YoloV3 jsou údaje o rychlosti spíše orientační, protože framework Darknet je obecně pomalejší než Tensorflow při použití stejného modelu.



Obrázek 5.3: Úspěšnost detektoru RetinaNet a TinyYolo nad testovacími daty HollywoodHeads.



Obrázek 5.4: Porovnání rychlosti modelů TinyYolo (TF), RetinaNet (TF) a YoloV3 (Darknet).

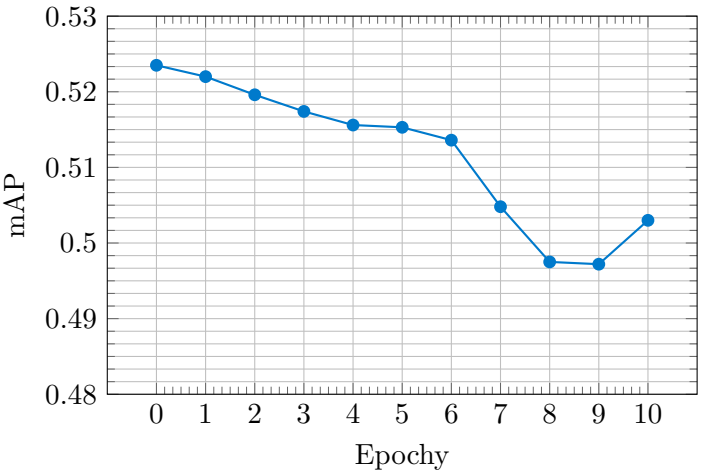
5.2 Úspěšnost učení ze získaných detekcí

Dataset pro otestování metody učení z vlastních anotací byl získán manuální anotací záběrů ze stejné bezpečnostní kamery, ze které se detektor sám doučoval. V trénovací a testovací

části videa se nevyskyjí stejné osoby. Po vyřazení záběrů, na kterých se žádní chodci nevyskytují, zůstalo v testovacím datasetu 124 anotovaných snímků.

Nad testovacími daty byl nejprve vyhodnocen obecně naučený model z datasetu HollywoodHeads a následně bylo otestováno deset snapshotů, které byly ukládány po dokončení epochy při dotrénování. Obecně natrénovaný model dosáhl 52,35 % mAp. Při testování snapshotů byla přesnost detekcí stagnující, po opakovaných iteracích byla patrná snižující se přesnost.

Možných příčin neschopnosti modelu zpřesnit detekce z vlastních anotací je několik. Na vině může být horší kvalita strojově získaných anotací, ze kterých se detektor učil. Při manuální kontrole bylo patrné, že některé boxy sice obsahovaly lidskou hlavu, ale jejich pozice byla nepřesná, což mohlo způsobit nižší hodnoty IoU. Další příčinou může být relativně malá velikost datasetu. V pozdějších epochách je pak model přeučený na jednotlivé snímky z datasetu. Úspěšnost modelu po jednotlivých epochách je zobrazena v grafu 5.5.



Obrázek 5.5: Úspěšnost detektoru RetinaNet na testovacích datech z bezpečnostní kamery. Na ose x je zobrazena hodnota mAP po jednotlivých epochách při učení z vlastních anotací. První hodnota v grafu je úspěšnost obecně natrénovaného modelu, která byla naměřena ještě před zahajením trénování.

5.3 Detekce hlavy na konkrétních ukázkách

Pro demonstraci detektorů na různých typech fotografií byly vybrány vhodné ukázky z testovacích částí datasetů HollywoodHeads 4.1.1 a Google Open Images 4.1.2. Ukázky zachycují detekce na rozličných fotografiích, především byl kladen důraz na případy, kdy měly detektory problémy s určitým typem scény. Zde zobrazené detekce byly naměřeny s jistotou detekce vyšší než 0,5.

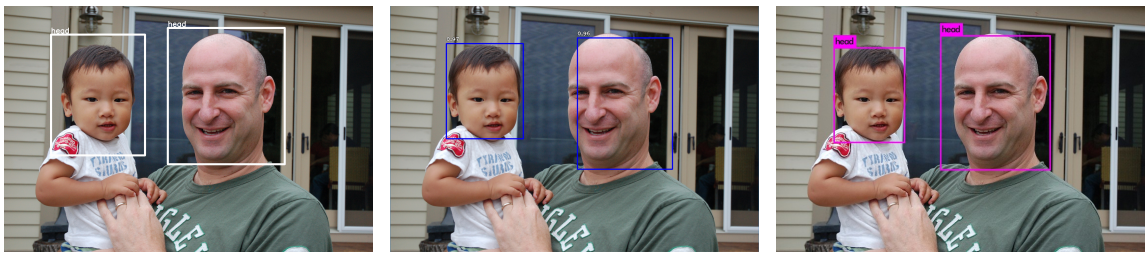
První ukázka (obrázek 5.6) zobrazuje detekci jediné hlavy v záběru na jednoduchém pozadí. Následně jsou zachyceny detekce více hlav v jednom snímku (obrázek 5.7) a detekce hlavy s různým natočením (obrázek 5.8). V posledních dvou ukázkách jsou scény s velkým počtem osob. U takto složité scéně (obrázek 5.9) se začínají projevovat nedostatky modelu Tiny-Yolo. Za povšimnutí stojí neoznačené hlavy v levém spodním rohu fotografie a dvojité

označení stejné hlavy v zadních řadách. Modely RetinaNet a YoloV3 měly u této fotografie detekce přesnější.

Úplně na závěr je pak ukázka detekce davu lidí s pokrývkami hlavy ve složité scéně (obrázek 5.10). Model Tiny-Yolo měl v tomto snímku pouze jednu úspěšnou detekci, modely RetinaNet a YoloV3 správně detekovaly pouze několik případů. Důvodem špatných detekcí v této fotografii je zřejmě kombinace pokrývek hlavy, slunečních brýlí a překryvů jednotlivých hlav. Podobné snímky se v trénovacích datech nevyskytovaly.

Tiny-Yolo RetinaNet YoloV3

Obrázek 5.6: Detekce jedné hlavy na jednoduchém pozadí. U fotografií tohoto typu neměly modely problém určit správně detekce s vysokou přesností.

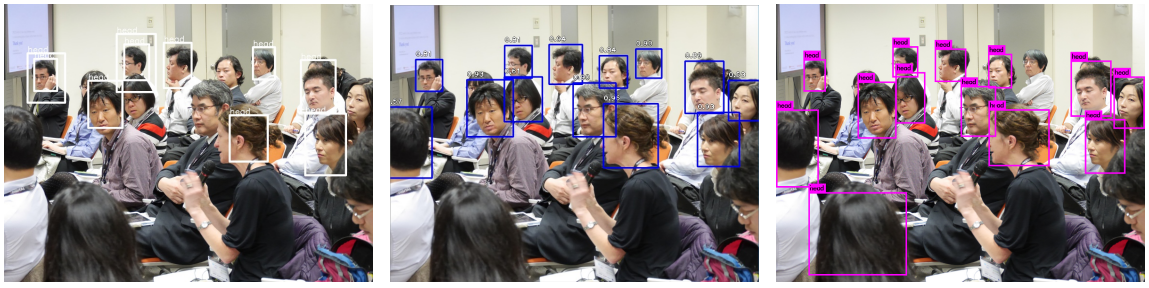


Tiny-Yolo RetinaNet YoloV3

Obrázek 5.7: Detekce několika hlav v jedné fotografii byly také vysoce úspěšné, zejména pokud se jednotlivé hlavy nepřekrývaly.

Tiny-Yolo RetinaNet YoloV3

Obrázek 5.8: Detekce hlavy s různým natočením.



Tiny-Yolo RetinaNet YoloV3

Obrázek 5.9: Detekce velkého počtu hlav ve složité scéně.

Tiny-Yolo RetinaNet YoloV3

Obrázek 5.10: Ukázka detekcí davu vzájemně překrývajících se osob.

Kapitola 6

Závěr

Tato práce se zabývá detekcí hlavy v obraze pomocí detektorů založených na neuronových sítích. Je zde popsán teoretický základ k pochopení fungování neuronových sítí a současné *state-of-the-art* klasifikační a detekční modely. Výsledný program je založen na modelu RetinaNet, který dosáhl vyšší přesnosti než ostatní testované modely. Aplikace umožňuje provádět detekce hlav na jednotlivých snímcích, videu nebo webkameře. Program je napsaný v jazyce Python, síť samotná je vytvořena pomocí frameworku Keras a knihovny Tensorflow.

Detekční modely byly natrénovány na anotovaném datasetu a jejich přesnost byla měřena metrikou mAP ze soutěží Pascal Visual Object Classes Challenge. RetinaNet s páteřní sítí ResNet dosáhl přesnosti 85,15 % a byl schopný provádět úspěšné detekce i v náročných scénách a dosahuje výrazně lepších výsledků než tradiční algoritmy, které pro detekci objektů nevyužívají neuronových sítí. U několika vybraných fotografií bylo také provedeno subjektivní srovnání mezi jednotlivými natrénovanými detektory.

V této práci byla také zkoumána schopnost detektoru dotrénovat se z vlastních anotací na snímcích ze scény se statickou kamerou. Tato metoda nevedla ke zlepšení přesnosti detektoru oproti obecně natrénovanému modelu. Podobný typ trénování je v praxi využíván pro některé úlohy (viz *semi-supervised learning*) a ačkoliv jeho implementace při detekci objektů nevedla k dobrým výsledkům, jedná se o zajímavou oblast výzkumu. Podobný přístup by bylo dobré vyzkoušet nejprve na klasifikačních úlohách, které jsou obecně jednodušší. U nich by nemohlo dojít k případu, že některé objekty v obrázku byly zaznačeny a jiné ne. Klasifikační síť by buď snímku přiřadila daný label nebo by byl snímek z datasetu vyřazen a přesnost anotací by tak šla garantovat použitím vysoké hodnoty *threshold*.

Využití neuronových sítí přináší mnoho možností budoucího rozšíření. K detečnímu systému by mohl být vyvinut algoritmus pro rozpoznání jednotlivých osob ve videu a sledování jejich pohybu ve snímaném prostoru s možným využitím většího počtu kamer snímajících prostor z různých úhlů. Oblast neuronových sítí je prudce se rozvíjející obor a je tedy možné, že přesnost zde využitých modelů bude brzy překonána dokonalejšími systémy, které by dosáhly srovnatelných nebo lepších detekcí a zároveň nižší výpočetní náročnosti.

Literatura

[1] Dalal, N.; Triggs, B.: Histograms of Oriented Gradients for Human Detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, Washington, DC, USA: IEEE Computer Society, 2005, ISBN 0-7695-2372-2, s. 886–893, doi:10.1109/CVPR.2005.177, [online].
URL http://dx.doi.org/10.1109/CVPR.2005.177

[2] Everingham, M.; Van Gool, L.; Williams, C. K. I.; aj.: The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, ročník 88, č. 2, Červen 2010: s. 303–338, [online].
URL http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.pdf

[3] Fukushima, K.: Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, ročník 36, 02 1980: s. 193–202, [online].
URL http://www.rctn.org/bruno/public/papers/Fukushima1980.pdf

[4] Girshick, R. B.: Fast R-CNN. *CoRR*, ročník abs/1504.08083, 2015, [online], 1504.08083.
URL http://arxiv.org/abs/1504.08083

[5] Girshick, R. B.; Jeff Donahue, T. D.; Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, ročník abs/1311.2524, 2013, [online], 1311.2524.
URL http://arxiv.org/abs/1311.2524

[6] He, K.; Gkioxari, G.; Dollár, P.; aj.: Mask R-CNN. *CoRR*, ročník abs/1703.06870, 2017, [online], 1703.06870.
URL http://arxiv.org/abs/1703.06870

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

[8] Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; aj.: Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, ročník abs/1207.0580, 2012, [online], 1207.0580.
URL http://arxiv.org/abs/1207.0580

[9] Hochreiter, S.: The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, ročník 6, č. 2, Duben 1998: s. 107–116, ISSN 0218-4885, doi:10.1142/S0218488598000094, [online].
URL http://dx.doi.org/10.1142/S0218488598000094

[10] Howard, A. G.; Zhu, M.; Chen, B.; aj.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR*, ročník abs/1704.04861, 2017.

[11] Ian Goodfellow, Y. B.; Courville, A.: Deep Learning, 2016, book in preparation for MIT Press, [online].
URL http://www.deeplearningbook.org

[12] Ioffe, S.; Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015, s. 448–456, [online].
URL http://jmlr.org/proceedings/papers/v37/ioffe15.pdf

[13] Karpathy, A.: What I learned from competing against a ConvNet on ImageNet. [online], [cit. 2017-12-18].
URL http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

[14] Karpathy, A.: Convolutinal Neural Networks for Visual Recognition. 2016, [online], [cit. 2018-02-20].
URL http://cs231n.github.io/convolutional-networks/

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

[16] LeCun, Y.; Bengio, Y.; Hinton, G.: Deep learning. *Nature*, ročník 521, 2015, [online].
URL http://graveleylab.cam.uchc.edu/WebData/mduff/MEDS_6498_SPRING_2016/deep_learning_nature_2015.pdf

[17] Lin, T.; Goyal, P.; Girshick, R. B.; aj.: Focal Loss for Dense Object Detection. *CoRR*, ročník abs/1708.02002, 2017, [online], 1708.02002.
URL http://arxiv.org/abs/1708.02002

[18] Liu, W.; Anguelov, D.; Erhan, D.; aj.: SSD: Single Shot MultiBox Detector. *CoRR*, ročník abs/1512.02325, 2015, [online], 1512.02325.
URL http://arxiv.org/abs/1512.02325

[19] McConnell, R.: Method of and apparatus for pattern recognition. [online].
URL https://patents.google.com/patent/US4567610

[20] Mehrotra, K.; Mohan, C. K.; Ranka, S.: *Elements of Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1997, ISBN 0-262-13328-8.

[21] Mishkin, D.; Matas, J.: All you need is a good init. *CoRR*, ročník abs/1511.06422, 2015, [online], 1511.06422.
URL http://arxiv.org/abs/1511.06422

[22] Pascanu, R.; Mikolov, T.; Bengio, Y.: Understanding the exploding gradient problem. *CoRR*, ročník abs/1211.5063, 2012, [online], 1211.5063.
URL http://arxiv.org/abs/1211.5063

[23] Redmon, J.; Divvala, S. K.; Girshick, R. B.; aj.: You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, ročník abs/1506.02640, 2015, [online], 1506.02640.
URL http://arxiv.org/abs/1506.02640

[24] Redmon, J.; Farhadi, A.: YOLO9000: Better, Faster, Stronger. *CoRR*, ročník abs/1612.08242, 2016, [online], 1612.08242.
URL http://arxiv.org/abs/1612.08242

[25] Redmon, J.; Farhadi, A.: YOLOv3: An Incremental Improvement. *arXiv*, 2018, [online], [cit. 2018-04-10].
URL https://arxiv.org/abs/1804.02767

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

[27] Sermanet, P.; Eigen, D.; Zhang, X.; aj.: OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *CoRR*, ročník abs/1312.6229, 2013, [online], 1312.6229.
URL http://arxiv.org/abs/1312.6229

[28] Shlens, J.: Train your own image classifier with Inception in TensorFlow. 2016, [online], [cit. 2018-02-21].
URL https://research.googleblog.com/2016/03/train-your-own-image-classifier-with.html

[29] Simonyan, K.; Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, ročník abs/1409.1556, 2014, [online], 1409.1556.
URL http://arxiv.org/abs/1409.1556

[30] Szegedy, C.; Liu, W.; Jia, Y.; aj.: Going Deeper with Convolutions. *CoRR*, ročník abs/1409.4842, 2014, [online], 1409.4842.
URL http://arxiv.org/abs/1409.4842

[31] Turk, M.; Pentland, A.: Eigenfaces for Recognition. *J. Cognitive Neuroscience*, ročník 3, č. 1, Leden 1991: s. 71–86, ISSN 0898-929X, doi:10.1162/jocn.1991.3.1.71, [online].
URL http://dx.doi.org/10.1162/jocn.1991.3.1.71

[32] Velikovič, P.: 2D Convolution. [online], [cit. 2018-02-21].
URL https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution

[33] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, ročník 1, 2001, ISSN 1063-6919, s. I–511–I–518 vol.1, doi:10.1109/CVPR.2001.990517.

[34] Volná, E.: Neuronové sítě 1. 2008, [online], [cit. 2018-02-02].
URL http://www1.osu.cz/~volna/Neuronove_site_skripta.pdf

[35] Vu, T.; Osokin, A.; Laptev, I.: Context-aware CNNs for person head detection. *CoRR*, ročník abs/1511.07917, 2015, [online], 1511.07917.
URL http://arxiv.org/abs/1511.07917

[36] Weisstein, E.: Haar Function. [online], [cit. 2018-02-01].
URL http://mathworld.wolfram.com/HaarFunction.html

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

[38] Zeiler, M. D.; Fergus, R.: Visualizing and Understanding Convolutional Networks. *CoRR*, ročník abs/1311.2901, 2013, [online], 1311.2901.
URL http://arxiv.org/abs/1311.2901