Chytré ořezávání obrázků s využitím umělé inteligence

Bc. Patrik Dendis

Diplomová práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2018/2019

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Patrik Dendis**
Osobní číslo:	**A17808**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Chytré ořezávání obrázků s využitím umělé inteligence**
Téma anglicky:	**The Smart Cropping of Images by means of Artificial Intelligence**

Zásady pro vypracování:

1. **Proveďte rešerši existujících řešení pro rozpoznávání objektů v mediální knihovně.**
2. **Seznamte se s dalšími možnostmi rozpoznávání objektů.**
3. **Vyberte vhodné řešení pro rozpoznávání objektů.**
4. **Implementujte zvolené řešení pro obrázky DAM (mediální knihovna).**
5. **Vytvořte webovou aplikaci na ořezávání obrázků, která ořeže a přizpůsobí obrázek nalezeným objektům.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ZELINKA, Ivan.: Umělá inteligence I. Volume 1. Zlín, Vutium, Brno, 1998, 126 p. ISBN 80-214-1163-5.**
2. **ŠIKUDOVÁ, Elena.: Detekcia a rozpoznávanie objektov, Vutium, Brno, 2014, 400 p. ISBN 978-80-87925-06-5.**
3. **BÍLA, Jiří. Umělá inteligence a neuronové sítě v aplikacích. Praha: Vydavatelství ČVUT, 1996, 115 s. ISBN 80-01-01275-1.**
4. **SOLEM, Jan Erik. Programming computer vision with Python. Beijing: O'Reilly, 2012, xii, 247. ISBN 978-1-4493-1654-9.**
5. **DADHICH, Abhinav.: Practical Computer Vision, Packt Publishing, 2018, 234 p.ISBN 9781788297684**
6. **ROSEBROCK, Adrian. Deep learning for computer vision with Python. Místo vydání není známé: PyImageSearch, 2017, 3 svazky (330, 280, 319 stran). ISBN 978-1-986538138.**
7. **SARFRAZ, Muhammad. Computer Vision and Image Processing in Intelligent Systems and Multimedia Technologies. Hershey, PA: Information Science Reference, 2014, 1 online zdroj. Advances in computational intelligence and robotics (ACIR) book series. ISBN 9781306861502.**

Vedoucí diplomové práce: **doc. Ing. Zuzana Komínková Oplatková, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **3. prosince 2018**

Termín odevzdání diplomové práce: **15. května 2019**

Ve Zlíně dne 7. prosince 2018

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

1. beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
2. beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
3. byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
4. beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
5. beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
6. beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
7. beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

1. že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
2. že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Patrik Dendis, v. r.
podpis diplomanta

ABSTRAKT

Cieľom diplomovej práce je navrhnúť a implementovať vhodný systém na orezávanie obrázkov, ktorý bude slúžiť k vytváraniu náhľadov z daných vstupných obrázkov. Systém by mal rozoznať objekty na obrázkoch a následne podľa vhodnej techniky vybrať objekty alebo samostatný objekt k orezaniu. Následne vygenerované náhľady rôznych veľkostí sa použijú pri automatickom generovaní marketingových materiálov ako sú plagáty, billboardy, atď. Ako vhodné techniky na rozoznávanie a detekciu objektov sa javí umelá inteligencia a konkrétne umelé neurónové siete.

Kľúčové slová: umelá inteligencia, neurónové siete, detekcia objektov, konvolučné siete, darknet

ABSTRACT

The main goal of my diploma thesis is to design and implement a suitable solution for image cropping, which will be used to create thumbnails of given input images. The system should recognize and detect objects in the images and then select object or more objects for cropping and generating thumbnails. Generated thumbnails of different sizes will be used to automatically generate marketing materials such as posters, billboards, etc. Artificial intelligence and neural networks appear to be suitable techniques for the recognition and detection of objects.

Keywords: artificial intelligence, neural networks, object detection, convolution networks, darknet

Touto cestou by som sa rád poďakoval všetkým, ktorí mi pomohli pri písaní diplomovej práce. Hlavne by som sa chcel poďakovať vedúcej práce doc. Ing. Zuzane Komínkovej Oplatkovej, Ph.D. za vedenie, rady a pripomienky, ktoré mi poskytla pri vypracovaní diplomovej práce.

Prehlasujem, že odovzdaná verzia diplomovej práce a elektronicky nahraná verzia do IS/STAG sú totožné.

OBSAH

ÚVOD

Umelá inteligencia je v súčasnosti veľmi aktuálnou a dynamicky sa rozvíjajúcou oblasťou v informatike. Je to vedná disciplína, ktorá za pomoci strojov alebo počítačových systémov simuluje správanie ľudskej inteligencie. V počítačovej vede zasahuje umelá inteligencia do niekoľkých odborov medzi ktoré patria napríklad hry, rozpoznávanie obrazu, rozpoznávanie reči, riadenie robotov, spracovanie a analýza dát, triedenie vzorov, analýza a predpovedanie štatistických postupností atď. Aplikácii a možností rozšírenia každým dňom pribúda.

Popri rozvoji umelej inteligencie kontinuálne narastá aj možnosť automatizácie niektorých ľudských činnosti. V nedávnej dobe sa na výrobu marketingových materiálov podieľal celý grafický tím. S príchodom automatizácie už odpadá nutnosť manuálneho nastavovania jednotlivých textových odstavcov pri marketingových plagátoch. O všetko nastavenie sa postará naprogramovaná aplikácia, ktorá je používateľsky prívetivá a jednoduchá.

Ako programátor a používateľ daných aplikácii sa stretávam pri problémoch hlavne s obrázkami. V prípade ak chce používateľ nastaviť nový obrázok na plagát, musí nastaviť jeho pozíciu manuálne. Tento proces je veľmi zdĺhavý a neefektívny. Ak chcú dané služby, poskytujúce vytváranie a úpravu marketingových materiálov konkurovať konkurencii, musia držať krok čo sa týka technologického napredovania. Preto som sa snažil prísť na to ako zautomatizovať generovanie relevantných náhľadov z obrázkov ľubovoľných rozmerov. Dané náhľady sa ďalej používajú napríklad pri vytváraní bilboardov alebo už spomínaných plagátov. V súčasnom riešení sa náhľady generujú a prispôsobujú podľa stredového bodu. Toto riešení spôsobuje, že je odrezaná alebo zanedbaná celá podstata obrázku a pri vytváraní plagátu s daným obrázkom je výsledok nedostatočný a je nutné manuálne prispôsobiť obrázok. Preto som sa rozhodol použiť a využiť silu umelej inteligencie pri spracovaní a analýze obrazu.

V prvej kapitole sa čitateľ zoznámi s mediálnou knižnicou DAM a jej vlastnosťami. DAM už obsahuje prvky automatického spracovania obrazu, ale pre daný prípad generovania náhľadov a použiteľnosti pri vytváraní marketingových materiálov je nedostatočný. Preto som sa ho rozhodol rozšíriť a vylepšiť danú funkcionalitu.

V ďalších kapitolách bude popísaný princíp umelej inteligencie, neurónových sietí a ich použitie. Detailne sa pozriem na konvolučné siete a ich použitie pri spracovaní obrazu. Postupne prejdem na metódy detekcie objektov, ktoré vyžívajú princíp umelej inteligencie a konvolučných sietí.

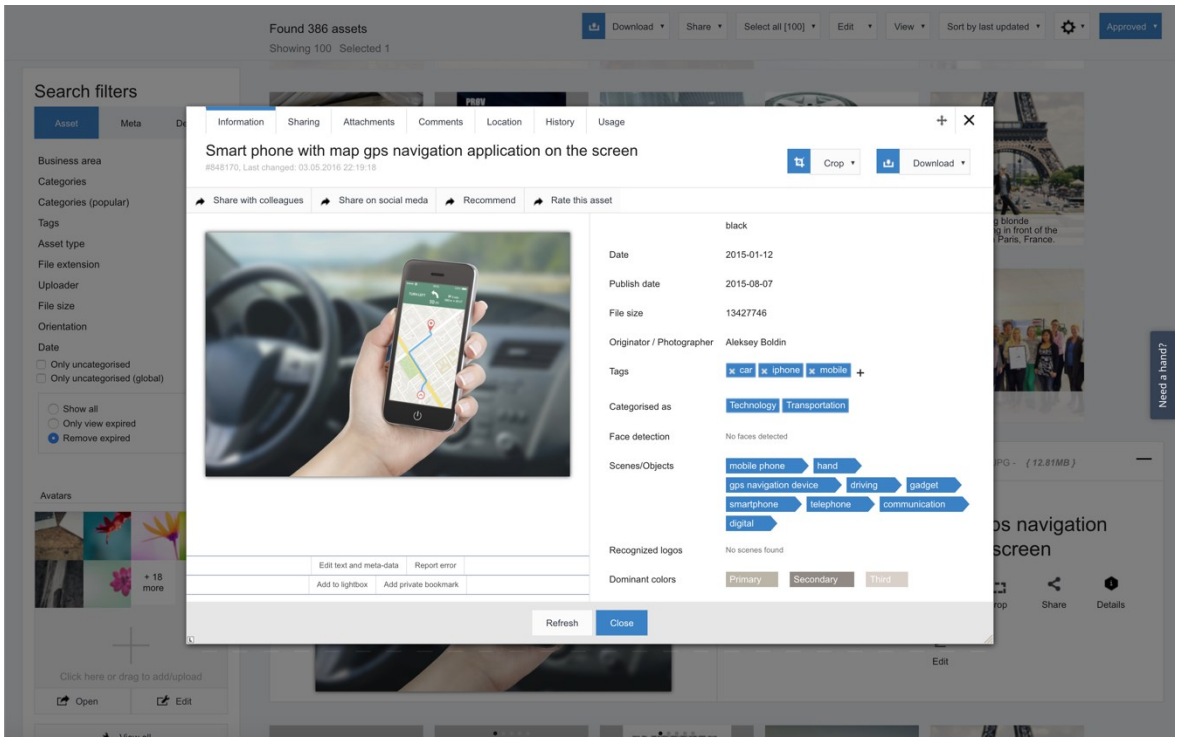
V štvrtej kapitole sa pozriem na najznámejšiu knižnicu pri spracovaní obrazu OpenCV a jej modul na podporu hlbokých neurónových sietí. Ďalej v tejto kapitole ukážem a opíšem frameworky, ktoré sa používajú pri práci s neurónovými sieťami medzi ktoré patria TensorFlow alebo Darknet.

Následne využijem nadobudnuté vedomosti z predchádzajúcich kapitol a navrhnem riešenie na orezávanie a generovanie náhľadov z obrázkov pomocou detekcie objektov a umelej inteligencie. V poslednom kroku dané riešenie implementujem do mediálnej knižnice DAM a ukážem jeho použitie pri generovaní marketingových materiálov.

I. TEORETICKÁ ČASŤ

1 MEDIÁLNA KNIŽNICA – DAM

Mediálna knižnica zhromažďuje, organizuje a umožňuje zdieľanie marketingových materiálov rôznych spoločností. Knižnica je produktom firmy Brandmaster v ktorej pracujem ako programátor. Všetky materiály použité pri výrobe plagátov, bilboardov alebo ostatných marketingových produktov sú uchovávané na jednom mieste. Používateľ mediálnej knižnice má možnosť ich organizovať do adresárov. Ponúka aj možnosť nastavovania prístupu pre jednotlivé materiály určené zamestnancom, prispievateľom, čitateľom alebo len administrátorom. DAM ponúka rozsiahly filter materiálov pomocou kategórii, formátu, veľkosti súboru, autora a ďalších možností. Pri nahrávaní nových materiálov sa automaticky vykonáva analýza nahrávaného súboru s cieľom identifikovať typ súboru, textový obsah, obsah obrázka, orientáciu, veľkosť súboru, ľudí, farbu, polohu, dátum, scenériu a úplné XMP o obrázku. XMP je štandard, pôvodne vytvorený spoločnosťou Adobe Systems Inc., na vytváranie, spracovanie a výmenu štandardizovaných a vlastných metadát pre digitálne dokumenty a súbory údajov [28]. Na rozpoznávanie tvári a objektov na obrázku DAM využíva platenú službu Google Vision API [1].



Obr. 1. Ukážka z mediálnej knižnice DAM [1].

1.1 Google Vision API

Rozhranie API služby Google Vision umožňuje vývojárom porozumieť obsah obrázka pomocou výkonných modelov strojového učenia v ľahko použiteľnom rozhraní REST API. Typická požiadavka na API obsahuje ako vstup dáta obrázkov, na ktorých sa má vykonať detekcia a súbor operácii, ktoré sa majú spúšťať na každom obrázkom.

Vision API je známe pre svoje presné výsledky a rozsiahlu dokumentácia obsahujúcu zbierku výukových programov, ktorá poskytuje veľmi podrobný prehľad o rozhraní API. Vývojári majú možnosť ľahko integrovať aplikácie z nasledovných funkcii.

- Optické rozpoznávanie znakov (OCR): Toto je klasický príklad počítačové videnia, ktoré sa primárne zaoberá extrakciou textu z obrazu.
- Detekcia vlastností obraz: Pomocou danej funkcii je možnosť získať všeobecné vlastnosti obrazu, farby a podobne.
- Detekcia objektov: Túto funkciu spolu s detekciou tvárí využíva aj mediálna knižnica a na jej výstupe je pole objektov, ktoré daná služba našla na vstupnom obrázku.
- Detekcia tvári: Úlohou danej funkcii je na obrázku alebo na množine obrázkov rozpoznať tváre, ktorú sú v nich prítomne. Využitie je pri systémoch sledovania.

Dané rozhranie a celý prístup služby Google Vision API vychádza z umelej inteligencie a neurónových sietí. Služba je veľmi jednoduchá a presná ale má dva zápory, ktoré rozhodli, že je lepšie použiť vlastné riešenie. Prvou nevýhodou je, že vývojár nemá kontrolu nad riešením a môže využívať len triedy objektov, ktoré poskytuje daná služba. Použitie Vision API je na princípe čiernej skrinky, kde vývojár vloží vstup a dostane výstup, ktorý nemôže nijak ovplyvniť. Druhou nevýhodou je cena služby. V prípade menej častého používania služby, čo je 1-1000 obrázkov za mesiac, je služba zadarmo. Pri mediálnej knižnici je nutnosť používať službu častejšie, kde sa cena za vstupný obrázkov pohybuje v rozsahu od 1$ až po 1.50$ [2].

1.1.1 REST API

Rest architektúra, ktorá sa najčastejšie používa pri vytváraní API (Application programming interface) rozhraní, využíva na komunikáciu HTTP protokol. Hypertext Transfer Protocol (Hypertextový prenosový protokol) je internetový protokol, ktorý pracuje na aplikačnej vrstve podľa modelu TCP/IP. Pôsobí ako primárna metóda na prepravu informácii vo webových aplikáciách. Pracuje na princípe request – response (požiadavka – odpoveď) medzi

klientom a serverom. Medzi metódy definované protokolom HTTP patria GET, POST, PUT, DELETE, HEAD, TRACE, OPTIONS a CONNECT [30].

REST je skratka pre Representational State Transfer a vytvoril ju v roku 2000 Roy Fielding. Rozhranie, využívajúce REST architektúru, je použiteľné pre jednotný a jednoduchý prístup ku zdrojom dát. Všetky zdroje dát majú vlastné identifikátory URI a REST, ktoré definujú základné metódy pre prístup k nim [31].

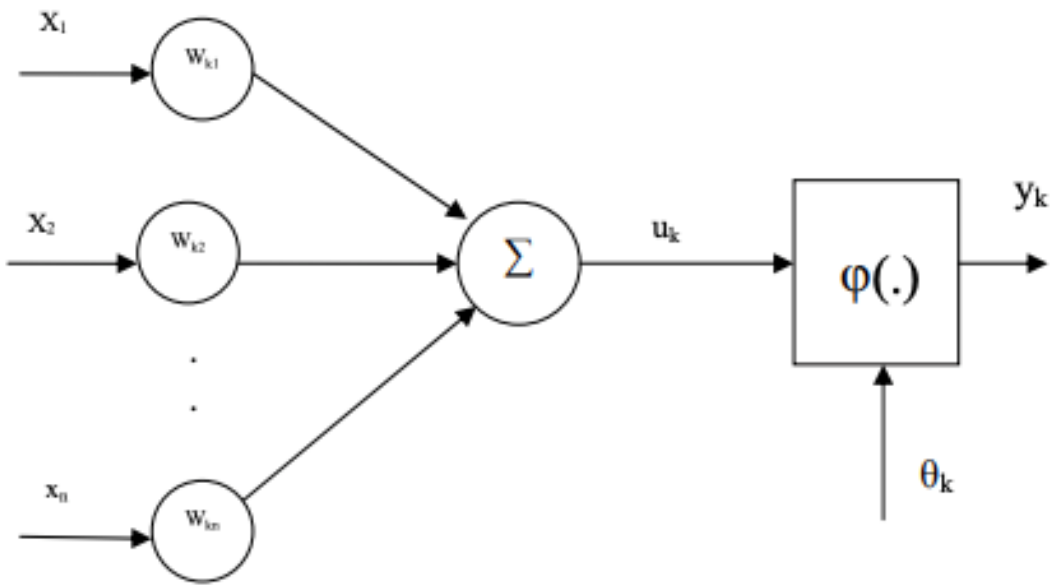
2 UMELÁ INTELIGENCIA

Ľudský mozog je vrcholom evolúcie na našej Zemi a zatiaľ najdokonalejším „nástrojom“ na spracovanie informácií. Čo je umelá inteligencia? Kedy môžeme povedať, že vytvorené zariadenie alebo aplikácia je inteligentná? Obecné testovanie umelej inteligencie je možné napríklad vo forme Turingovho testu, ktorý publikoval Alan Touring v roku 1950 vo svojom článku „Computing machinery and intelligence“. Tento test spočíva tom, že máme človeka, ktorý kladie otázky 2 respondentom, ktorých ale nevidí. Jeden z respondentov je počítač a druhý je človek. Konverzácia trvá 5 minút a rozsah tém nie je nijako ohraničený. Samozrejme konverzácia prebieha výlučne v textovej podobe. Pokiaľ osoba, ktorá kladie otázky nie je schopná rozlíšiť, ktorý z respondentov je počítač, môžeme ho považovať za inteligentný [3].

Existuje niekoľko druhov algoritmov a aplikácii, ktoré majú za účel napodobniť inteligentné správanie. Jednou z nich sú pravé umelé neurónové siete.

2.1 Umelé neurónové siete

Neurónové siete sú siete pospájaných uzlov, v našom prípade neurónov. Neuróny sú prepojené nasledovným spôsobom. Každý výstup z neurónu slúži ako vstup do ďalších neurónov. Princíp fungovania je inšpirovaný ľudskom mozgom. Ktorý obsahuje biologické neuróny a synapsie (spojenie medzi neurónmi). Sila ľudského mozgu je vo využití obrovského množstva "pomalých" neurónov, zoskupených do veľmi zložitej masívne paralelnej siete, ktorej veľkosť a topológia je v simulačných procesoch zatiaľ zďaleka nenapodobniteľná. Umelý neurón pozostáva z nasledujúcich základných častí viď obr. 2 [5].



Obr. 2. Model neurónu [6].

x_n – vstupy

w_{kn} – synaptické váhy pre vstup

Σ – sumačná funkcia

u_k – výstup sumačnej funkcie

φ(.) – aktivačná funkcia

yk – výstup neurónu

θ_k – prah neurónu(threshold)

Model na obrázku obsahuje aj prah neurónu, ktorý zabezpečuje zníženie (threshold) alebo zvýšenie (bias) hodnoty vstupu do aktivačnej funkcie. Matematický môžem opísať neurón nasledujúcimi rovnicami (1,2,3).

$$u(k) = \sum_{i=1}^{n} (w_{ki} * x_i) \tag{1}$$

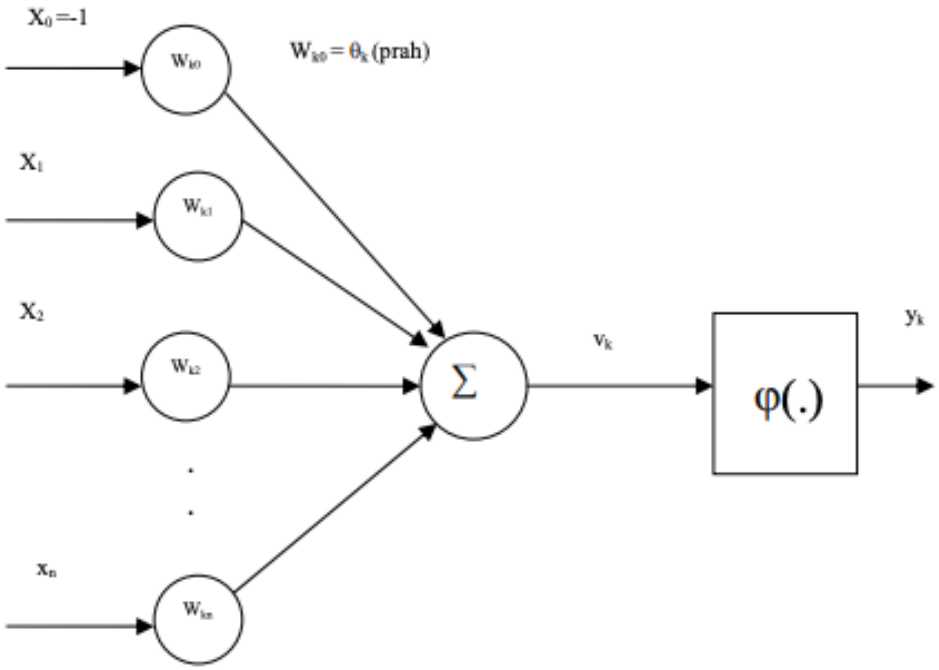
$$v(k) = u_k - \theta_k \tag{2}$$

$$y(k) = \varphi(v_k) \tag{3}$$

Z rovnice (1) dostávame výstup lineárneho kombinátora, ktorý predstavuje sumu vstupných hodnôt vynásobených ich príslušnou váhou. Použitím prahu θ_k v rovnici (2) dostávam vnútornú aktivitu neurónu k. Rovnice môžem zapísať aj iným spôsobom ak pridám vstup $x_0 = -1$ a váhu $w_{ko} = \theta_k$ [5].

$$v(k) = \sum_{i=0}^{n} (w_{ki} * x_i) \tag{4}$$

$$y(k) = \varphi(v_k) \tag{5}$$

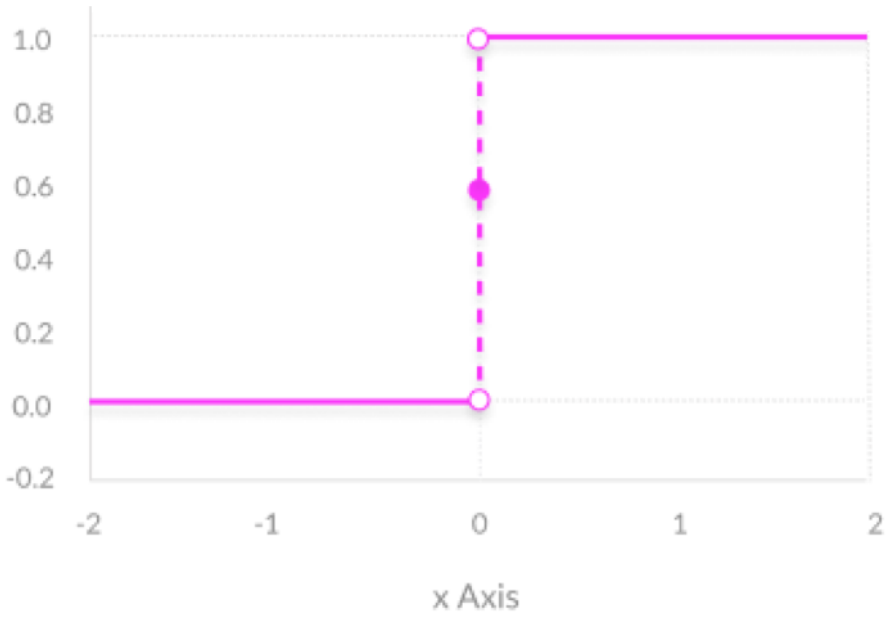


Obr. 3. Model neurónu s prahovou hodnotou -1 [5].

2.1.1 Aktivačná funkcia

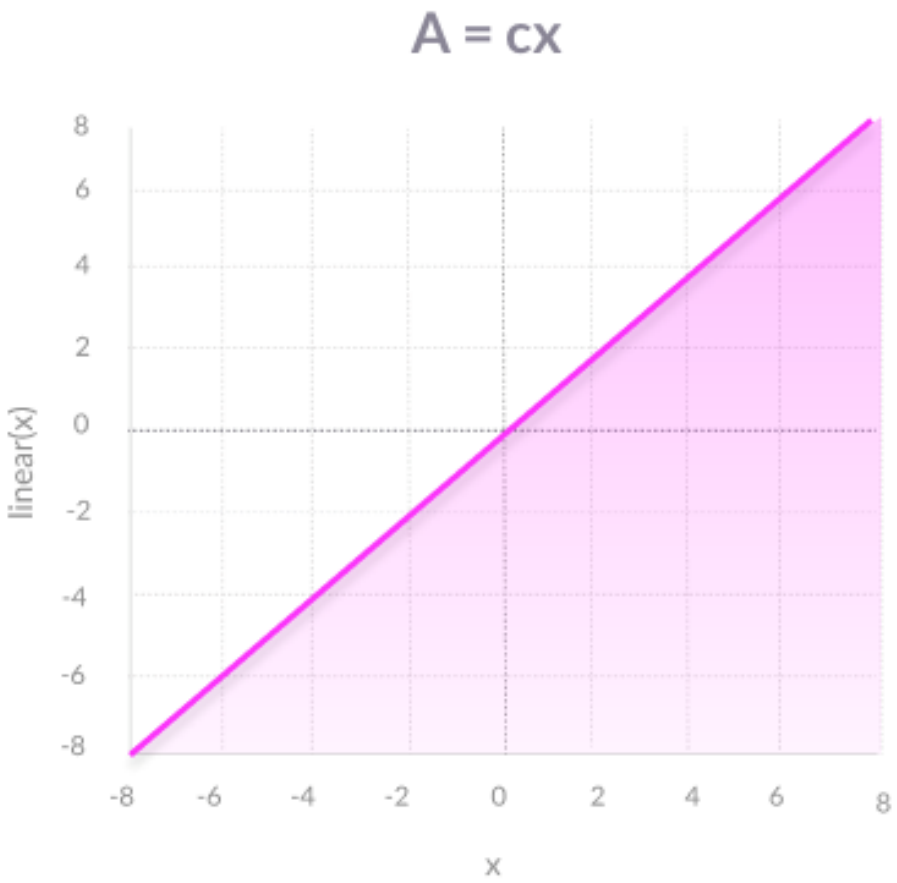
Cieľom aktivačnej, inak nazývaná aj prenosová, funkcie je ohraničiť výstup z neurónu. Typický normalizovaný rozsah výstupu je z interval <0,1> alebo <-1,1>. Existuje veľa prenosových funkcii ale všetky rozlišujeme do troch základných skupín: skokové, lineárne a nelineárne. Výber funkcie je podmienený aký problém neurónová sieť rieši. Najčastejšie používané aktivačné funkcie sú nelineárne funkcie. Funkcie sú ostro rastúce funkcie, ktoré predstavujú rovnováhu medzi lineárnym a nelineárnym správaním funkcie [6].

Skoková prenosová funkcia je podmienená prahu neurónu. Ak je vstupná hodnota väčšia alebo menšia ako prah, neurón je aktivovaný a na výstupe dostávame hodnoty [0,1]. Hlavný problém danej funkcie je, že na výstupe dostaneme len binárnu hodnotu [6].



Obr. 4. Skoková prenosová funkcia.

Po častiach lineárne funkcie vytvárajú výstup, ktorý je úmerný vstupným signálom ale umožňuje viac výstupov, nie len áno a nie. Problém nastáva ak by sme chceli trénovať daný model pomocou backpropagation. Pretože derivácia danej funkcie je konštanta a nemá žiadny vzťah k danému vstupu. Tým pádom nie je možné sa vrátiť a identifikovať, ktoré váhy vstupného neurónu by po zmene dali lepší výsledok [6].

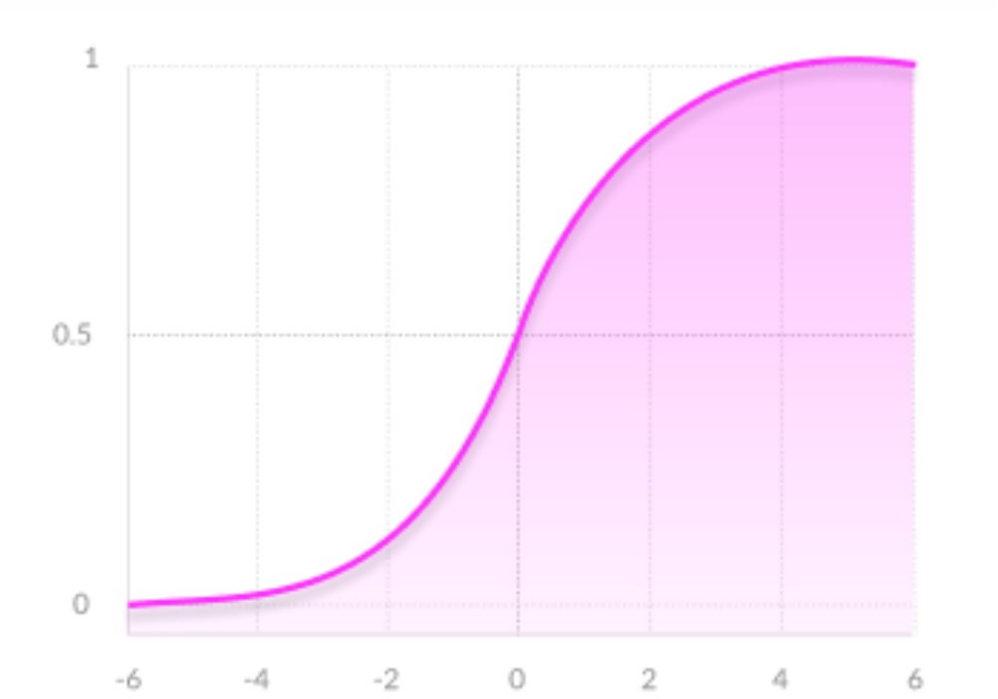


Obr. 5. Lineárna prenosová funkcia.

Moderné neurónové siete používajú nelineárne aktivačné funkcie, ktoré poskytujú vytvorenie komplexného prepojenia medzi vstupmi a výstupmi. Po derivácii funkcie nedostávame konštantu ale funkciu, ktorá ma prepojenie na vstup a tým je možné trénovanie a modelovanie dát napríklad obrázkov, videa, audia atď. Používa sa nasledujúca rovnica sigmoidnej funkcie (6).

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{6}$$

kde a je parameter strmosti a možno ním meniť tvar sigmoidnej funkcie. Je definovaná ako monotónne rastúca, hladká a ohraničená funkcia [6].

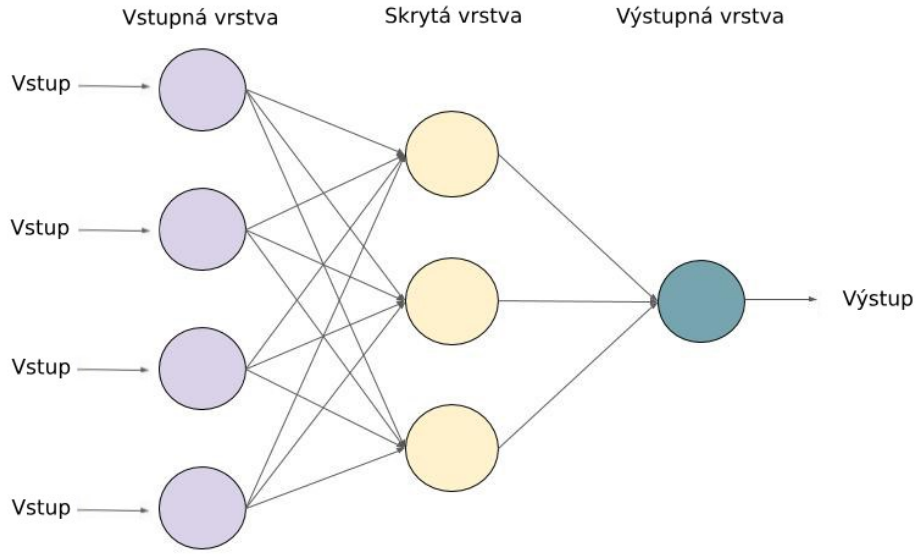


Obr. 6. Nelineárna sigmoidná funkcia.

2.1.2 Architektúra neurónových sietí

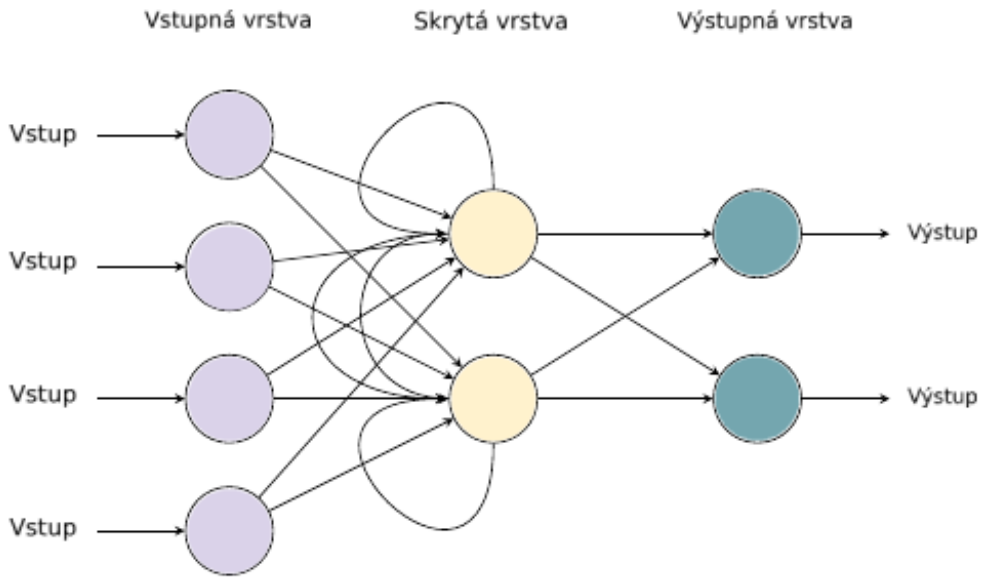
Pri riešení zložitejších problémov jeden neurón nestačí. Z tohto dôvodu sa neuróny spájajú do takzvaných neurónových sietí. Počet neurónov a ich vzájomné prepojenie sa nazýva architektúra alebo topológia neurónovej sieti. V sieti rozdeľujeme neuróny do troch základných kategórii. Prvú kategóriu tvoria vstupné neuróny. Na vstupné neuróny privádzame spravidla dáta. Napríklad jeden vstupný neurón môže predstavovať hodnotu odtieňa sivej farby pre jeden pixel. Druhú kategóriu tvoria skryté neuróny a poslednou kategóriou sú výstupné neuróny. Na výstupných neurónoch dostávame príslušné výsledky. Skryté neuróny sa nachádzajú medzi vstupnými a výstupnými neurónmi [7].

Stavy všetkých neurónov v sieti určujú stav neurónovej siete a váhy spojov určujú konfiguráciu neurónovej siete. Neurónová sieť sa v čase vyvíja, mení sa prepojenie a stav neurónov, adaptujú sa váhy. Existuje mnoho druhov neurónových sietí. Je ich však možné rozdeliť do dvoch základných kategórii. Rozdelenie podľa topológie siete je jedno z najčastejších, pričom základné sú dopredná (feedforward) a rekurentná (feedback, recurent). V doprednej neurónovej sieti sa nikdy nevytvára prepojenie medzi neurónmi z vyššej vrstvy s neurónmi z nižšej vrstvy. Tým pádom sa nevytvárajú cyklické prepojenia. Sú špecifické najmä svojou rýchlosťou. Je možné použiť širokú škálu konvenčných numerických metód (napríklad Levenberg – Marqardtova metóda) na trénovanie [7].



Obr. 7. Schéma doprednej neurónovej siete [7].

V prípade rekurentných neurónových sietí je možné vytvárať cykly. Nevýhodou niektorých sietí tohto typu je, že musí každý nový vstup iterovať dlhú dobu pred dosiahnutím žiadúceho výsledku. Preto rekurentné siete sa učia veľa ťažšie ako dopredné [7].



Obr. 8. Schéma rekuretnej neurónovej siete [7].

2.1.3 Učenie neurónových sietí

Učenie je proces, v ktorom sa menia parametre neurónových sietí v závislostí od nastavených pravidiel. Zmysel trénovania alebo učenia je zabezpečiť schopnosť produkovať

správne zodpovedajúce výstupy pre rôzne vstupné dáta. Pri trénovaní je veľmi dôležité, aby trénovácie dáta neboli príliš špecifické pretože to spôsobuje preučenie (overfitting). Preučenie je stav, v ktorom sieť produkuje veľmi dobré výsledky na vstupné dáta z trénovacej sady ale pre ostatné dáta je takmer nepoužiteľná. Učenie sa rozdeľuje do troch hlavných kategórii:

- učenie s učiteľom (supervised),
- učenie bez učiteľa (unsupervised),
- učenie s posilňovaním (reinforcement learning) [9].

Pri učení s učiteľom je sieť opakovane vystavovaná vstupom, ktoré majú očakávanú odpoveď. Príkladom učenia s učiteľom je učenie pomocou spätného šírenia chyby [9].

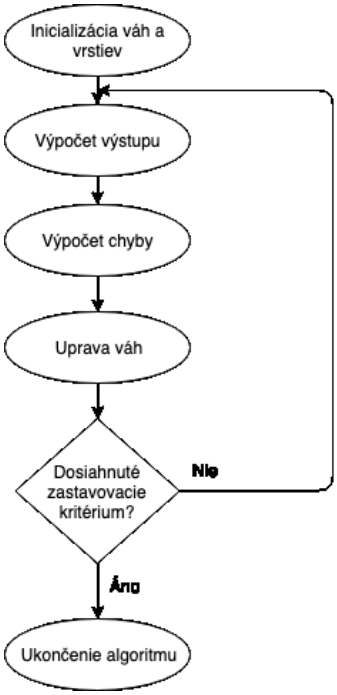
Trénovanie bez učiteľa (unsupervised) je odlišné v tom, že ku trénovacím vstupom nie je priradený výstup ale neurónová sieť musí sama odvodiť význam zo vstupných dát a extrahovať ich vlastnosti. Najčastejším typom trénovania je kompetitivné učenie, kde neuróny navzájom súťažia medzi sebou o to, kto vyprodukuje najlepšie výstupné dáta zo vstupných vzoriek. Príkladom učenia bez učiteľa sú samoorganizujúce mapy [9].

Pri trénovaní s posilňovaním sa sieť snaží namapovať rôzne situácie tak aby mala čo najväčšiu numerickú hodnotu. Sieť nevie aké akcie má vykonať ale sama musí objaviť, za ktoré akcie dostane najväčšiu numerickú hodnotu. Trénovanie je o postupnom rozhodovaní kde výstup závisí od aktuálneho vstupu a ďalší vstup závisí od predchádzajúceho vstupu [9].

Proces trénovania sa môže interpretovať ako problém optimalizácie, kde je požadované minimalizovať strednú kvadratickú chybu (MSE, Mean Square Error) na trénovacích dátach. Na riešenie problému sa používajú štandardné optimalizačné heuristiky ako napríklad simulované žíhanie alebo špecializované gradientové metódy. Pri neurónových sieťach sa najčastejšie využíva metóda učenia pomocou spätného šírenia chyby [8].

2.1.4 Algoritmus spätného šírenia chyby

Metóda učenia pomocou spätného šírenia chyby (error back-propagation) je jeden z najdôležitejších spôsobov trénovania viacvrstvových neurónových sietí. Ako to vyplýva z nazvú algoritmus pracuje spätne od výstupnej vrstvy k vstupnej a sieť musí byť plne prepojená. To znamená, že každá vrstva je plne prepojená s nasledujúcou vrstvou. Základným cieľom je minimalizovať celkovú chybu siete. Na obrázku je znázornený zjednodušený algoritmus spätného šírenia chyby [10].



Obr. 9. Diagram algoritmu spätného šírenia chyby.

1. inicializácia váh

Inicializácia váh je možná pomocou určitého algoritmu, ako napríklad Nguyen-Widrow algoritmus alebo náhodne. Pri náhodnej inicializácii sa používajú veľmi malé hodnoty v intervale <-1, 1>.

2. výpočet výstupu

Výpočet výstupu, alebo aj inak nazývaný forward pass, je vlastne výpočet výstupu siete. Vstup prechádza zo vstupnej až na výstupnú vrstvu.

3. výpočet chyby

V tomto kroku sa vypočítavajú chybu podľa ktorých sa ďalej používajú na úpravu váh. Najprv sú vypočítané chyby na jednotlivých neurónoch podľa vzťahu (7).

$$\varepsilon_i = d_i - y_i \tag{7}$$

Kde ε_i je chyba na konkrétnom neuróne, d_i je požadovaný výstup na neurónu a y_i je aktuálny výstup z neurónu. Hodnota ε_i sa ďalej používa na výpočet hodnoty δ_i. Táto hodnota sa počíta podľa nasledujúceho vzorca (8).

$$\delta_i = \varepsilon_i * f'(y_i) \tag{8}$$

Kde f' je derivácia aktivačnej funkcie neurónu. Hodnota δ_i je potrebná na výpočet hodnoty δ_j pre predchádzajúce vrstvy podľa pomocou vzťahu (9).

$$\delta_j = \alpha * f'(y_j) * \sum_{i=0}^{N} \delta_i * w_{ji} \tag{9}$$

V rovnici (9) je K hodnota, ktorá udáva počet neurónov v danej vrstve. Ďalší parameter je α, nazývaný aj rýchlosť učenia, ktorý rozhoduje o miere ako má byť váha w upravená.

4. úprava váh

Vypočítané hodnoty z predchádzajúceho kroku sú ďalej použité na výpočet rozdielu Δw_{ji}, ktorý sa vypočíta podľa vzťahu (10).

$$\Delta w_{ji} = \delta_j * y_i \tag{10}$$

Výsledok rozdielu pripočítame k aktuálnym váham. Týmto vzťahom dostávame novú hodnotu váh (11).

$$w_{ji} = w_{ji} + \Delta w_{ji} \tag{11}$$

5. zastavovací algoritmus

Algoritmus končí, ak je dosiahnuté zastavovacie kritérium, ktoré je zvyčajne vypočítané ako priemerná kvadratická chyba po každej iterácii na trénovacích dátach. Zastavovacia hodnota MSE je volená používateľom algoritmu. Niekedy sa používajú na zastavovanie algoritmu aj iné techniky ako napríklad použitie validačnej množiny dát [10].

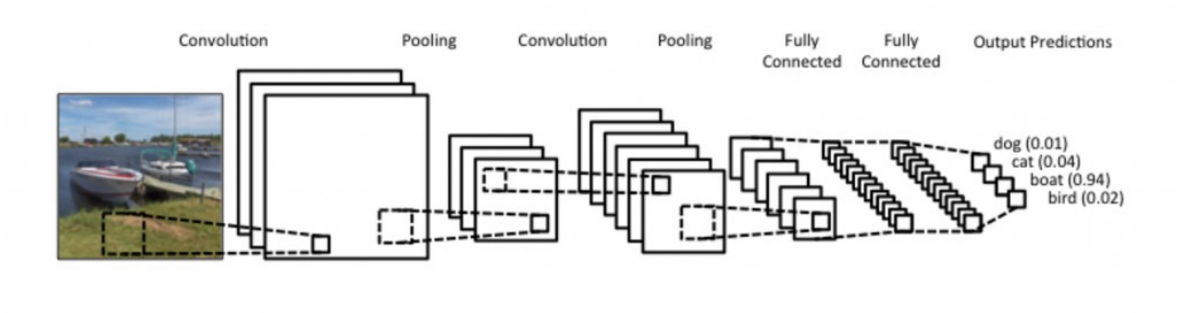
2.2 Konvolučné siete

Konvolučné neurónové siete (CNN) sú špeciálnym typom neurónových sietí. Vychádzajú z dopredných neurónových sietí a najčastejšie sa využívajú pri detekcii objektov na obrázkach alebo videách. Konvolučné siete vznikli kvôli tomu aby vylepšili dopredné neurónové siete, ktoré sa spájajú s niekoľkými problémami. Plne prepojené vrstvy s veľkým počtom neurónov môžu mať stovky tisíc váhových prepojení. Siete sa tak stávajú veľké, v procese učenia obsahujú veľa voľných parametrov a sú náchylné napríklad na posun znaku pri rozoznávaní objektov [11].

Konvolučné siete využívajú zdieľanie váh, čo zmenšuje počet voľných parametrov a zároveň zvyšuje robustnosť siete proti posunu. Ich základ je postavený na princípe neocognitronu,

ktorý predstavil Kunihiko Fukushima v roku 1980. Medzi ich hlavnú výhodu patrí, že sa vedia vysporiadať s určitou mierou posunu alebo deformáciou vstupných dát. Je to spôsobené hlavne získavaním príznakov z recepčných polí vstupných neurónov, zdieľaním dát a priestorovým vzorkovaním. Príznaky z recepčných polí nám pomáhajú získať základné príznaky, napríklad hrany alebo rohy. Tieto príznaky sú v ďalšej vrstvy kombinované a tým získam príznaky na vyššej úrovni. Zrýchli sa tým proces zložitého predspracovania a získavania príznakov [11].

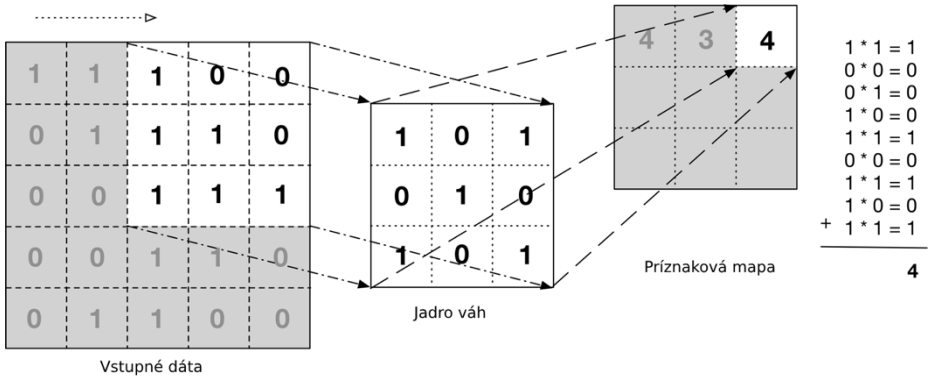
Na obr. 10 je znázornený typický príklad konvolučnej neurónovej siete. Na prvú konvolučnú vrstvu, ktorá sa skladá z konvolučných príznakových máp, je privedený vstupný obraz. Príznakové mapy medzi sebou zdieľajú svoje parametre. Výstup z vrstvy je privedený na vzorkovaciu vrstvu, ktorej cieľom je zmenšiť daný výstup. Tieto dvojice vrstiev sa striedajú a ich množstvo záleží od konkrétnej implementácie. Výstup z poslednej vzorkovacej vrstvy je ďalej privedený do plne prepojenej neurónovej siete [11].



Obr. 10. Príklad konvolučnej neurónovej siete [29].

Konvolučná sieť je tým pádom rozdelená na konvolučnú časť a výstupnú časť. Konvolučná časť obsahuje konvolučné a vzorkovacie vrstvy a výstupná časť je klasický viacvrstvový perceptrón [11].

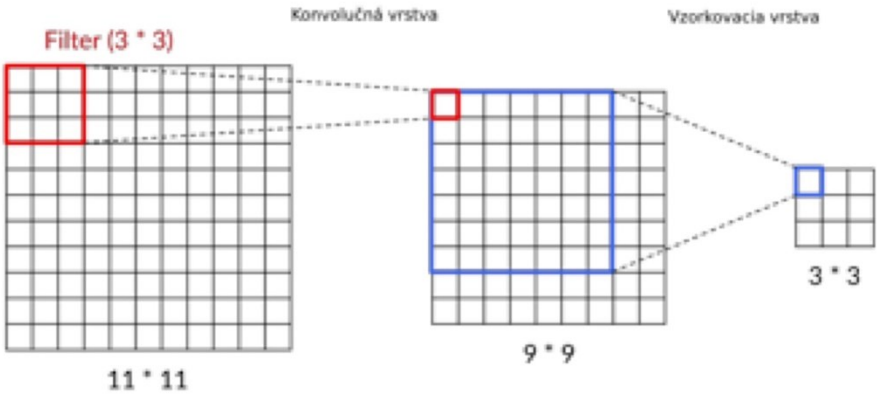
Každý neurón v konvolučnej vrstve je prepojený s nejakým malým okolím k spracovanému obrázku na zodpovedajúcich súradniciach, ktoré tvoria mapu. Každá príznaková mapa obsahuje vlastné zdieľané jadro váh pre prepojenú mapu z nižšej vrstvy. Zdieľanie váh je veľmi prospešné na výpočtovú zložitosť v procese učenia a taktiež na pamäťovú náročnosť [8].



Obr. 11. Konvolúcia medzi mapami [8].

Na obr. 11 je znázornená konvolúcia medzi konvolučnou mapou a mapou predchádzajúcej vrstvy. Je to základná funkcia, ktorú neuróny v týchto vrstvách počítajú. Vstupné dáta si môžeme predstaviť ako obrázok o veľkosti 5x5. Vstupné dáta sú vynásobené s váhami jadra 3x3. Na výstupe dostávame príznakovú mapu. Jadro váh sa nastavuje v procese učenia. [8]

Vzorkovacia vrstva obsahuje rovnaký počet príznakových máp ako v predchádzajúcej konvolučnej vrstve. Hlavným cieľom vzorkovacej vrstvy je redukovať priestorové rozlíšenie príznakových máp. Rozdielom medzi konvolučnou a vzorkovacou vrstvou je v prepojeniach s nižšou vrstvou. V prípade konvolučnej mapy je jadro váh aplikované na skupiny neurónov, ktoré sa posúvajú po jednom a spôsobuje to vzájomné prekrývanie. Vzorkovacia mapa aplikuje jadro na vzájomne nezávisle skupiny neurónov [8].



Obr. 12. Ukážka konvolúcie a vzorkovania [8].

2.2.1 Učenie konvolučných sietí

Najbežnejším učiacim algoritmom pre učenie konvolučných sietí je algoritmus spätného šírenia chyby, ktorý je takmer totožný ako som popísal pre štandardnú viacvrstvovú sieť perceptronov. Jediný rozdiel je v tom, že treba brať do úvahy zdieľanie váh v konvolučných a vzorkovacích vrstvách. Každý neurón má svoje prepojenie s nižšou vrstvou a neuróny zdieľajú váhy a bias, čím je značne znížený počet trénovaných parametrov. Vzorkovacie vrstvy majú len jednu váha a jeden trénovateľný bias. Malý počet parametrov, ktoré sa upravujú, vedie k významne menšej potreby výpočtového výkonu pri trénovaní. Konvolučné siete sú veľmi vhodné a často používané na implicitnú extrakciu vlastnosti pri istej miere skreslenia. Preto sa používajú v klasifikačných úlohách, najmä pri rozpoznávaní vzorov, písaných písmen, číslic, iných znakov alebo pri rozpoznávaní objektov, tvári na obrázku [8].

3 DETEKOVANIE OBJEKTOV

Cieľom detekcie objektov je identifikovať všetky objekty zo známej alebo naučenej triedy ako napríklad ľudí, autá alebo tváre. Typicky je na obraze len malý počet objektov ale existuje veľké množstvo možných umiestnení, ktoré treba nejakým spôsobom nájsť. Každá detekcia objektu obsahuje nejakú formu informácie o polohe daného objektu. Najčastejšie sa používajú súradnice ohraničeného rámčeku (bounding box). V niektorých situáciách je informácia o polohe detailnejšia a obsahuje parametre lineárnej alebo nelineárnej transformácie. Napríklad detektor tváre môže počítať aj súradnice očí, nosa alebo úst. Pozícia môže byť tiež definovaná trojrozmernou transformáciou, ktorá špecifikuje umiestnenie objektu vzhľadom na kameru. Systémy na detekciu objektov vytvárajú model z tréningového súboru. Pre jeden daný objekt je potreba veľké množstvo rovnakých alebo podobných trénovacích objektov jedného druhu. Dôvodom je aby model bol prispôsobený na variabilitu [8].

Veľmi často sa zamieňa detekcia s rozpoznávaním objektov. Metódy rozpoznávania objektov vracajú na výstupe len popis objektu a nie informácie o polohe daného objektu.

Obr. 13. Rozdiel medzi rozpoznávaním a detekovaním objektov.

Metódy detekcie objektov zaraďujeme do dvoch hlavných kategórii podľa prístupu na metódy založené na strojovom učení a metódy založené na hlbokom učení. Pre metódy strojového učenia najprv treba definovať vlastnosti (features) a potom pomocou techniky ako je Support Vector Machine (SVM) sa vykoná klasifikácia. Pri metódach z druhej kategórie je možné robiť detekciu objektov bez špecifikovania vlastnosti. Metódy založené na hlbokom učení typicky využívajú silu konvolučných neurónových sietí [12].

Medzi metódy prvej kategórie patria napríklad:

- Viola-Jones,

- SIFT (Scale-invariant feature transform),
- HOG (Histogram of oriented gradients).

Pre moje účely som sa rozhodol použiť a popísať metódy založené na hlbokom učení, medzí ktoré patrí:

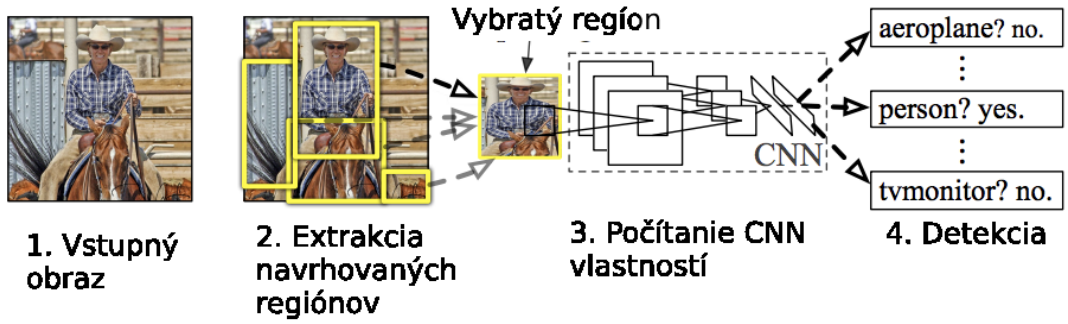
- R-CNN,
- Fast R-CNN,
- Faster R-CNN,
- SSD,
- YOLO [12].

3.1 Metódy založené na hlbokom učení

3.1.1 R-CNN

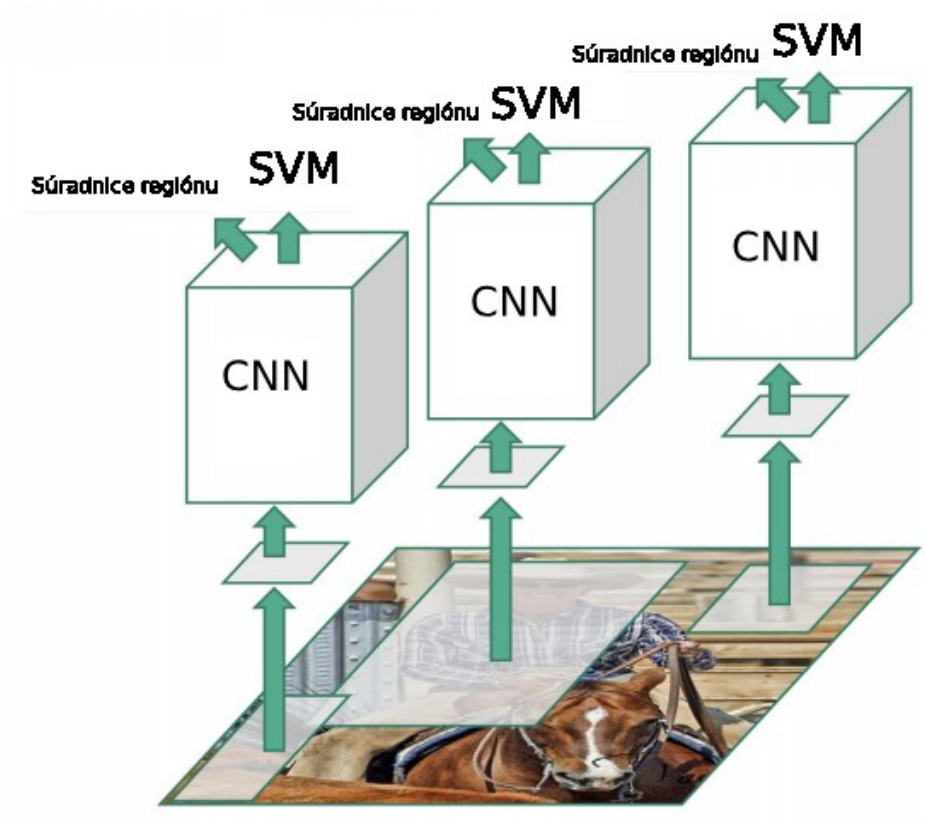
R-CNN spolu s Fast R-CNN a Faster R-CNN patria medzi metódy založené na navrhovaných regiónoch. R-CNN metódu prvý krát predstavili v roku 2014 Ross Girshick, Jeff Donahue, Trevor Darrell a Jitendra Malik. Podstatou celej metódy je výber 2000 regiónov, ktoré sa vyberajú pomocou selektívneho vyhľadávacieho algoritmu. Teda namiesto toho aby sa klasifikovalo obrovské množstvo regiónov, algoritmus pracuje len s 2000 regiónmi. Selektívny algoritmus pracuje v nasledujúcich troch krokoch:

1. Generovanie počiatočnej subsegmentácie, ktorá vytvorí množstvo kandidátskych regiónov.
2. Následne sa použije Greedy algoritmus, ktorý rekurzívne kombinuje podobné regióny a zlučuje ich do väčších regiónov.
3. V poslednom kroku sa z generovaných regiónov vytvoria navrhované regióny [13].



Obr. 14. Schéma R-CNN metódy [13].

Týchto 2000 navrhovaných regiónov je následne upravených na štvorcový tvar a každý z regiónu je vložený do konvolučnej neurónovej siete. Výstup z konvolučnej siete je 4096 dimenzionálny vektor. Konvulučná sieť sa stará o extrakciu vlastností a vzorov, ktoré sú v poslednom kroku pomocou algoritmu SVM (Support Vector Machine) klasifikované do jednotlivých naučených tried [13].



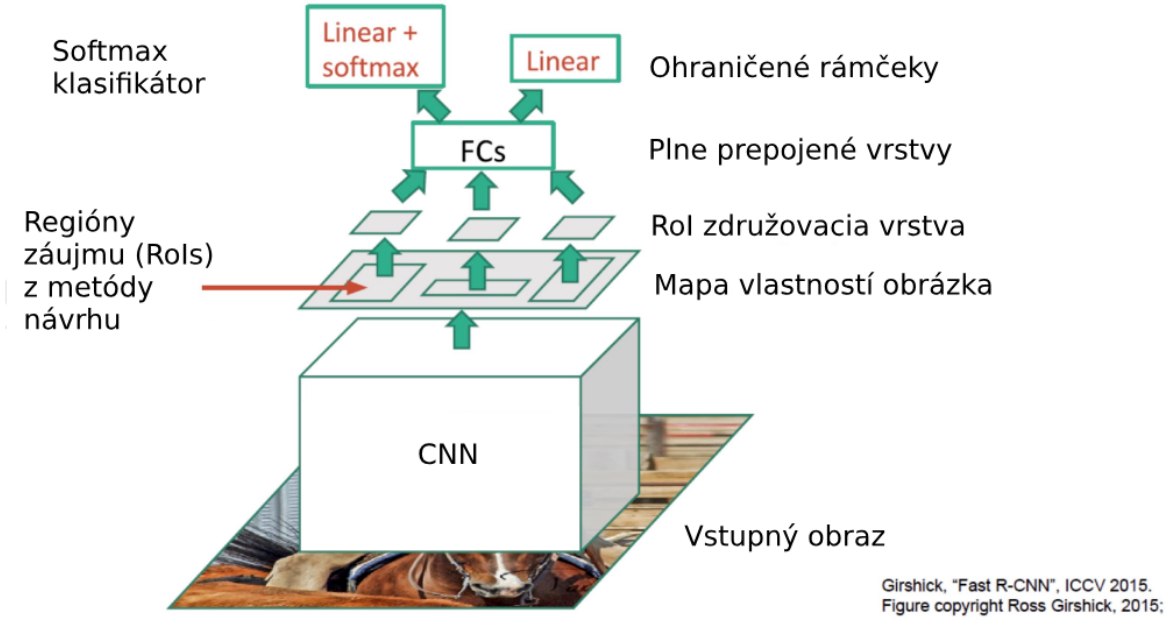
Obr. 15. Schéma R-CNN s implementovaným SVM algoritmom [13].

R-CNN pracuje dobre ale nie je vhodný pre detekovanie objektov v reálnom čase, pretože každý jeden navrhovaný región sa musí poslať cez konvulučné siete. Tým pádom na jeden obrázok potrebujeme urobiť CNN analýzu 2000 krát. Trénovanie pri danej metódy je veľmi náročné, pretože sa musia trénovať tri rôzne modely separátne – konvolučná sieť na generovanie vlastností, klasifikátor na predikovanie tried a regresný model na zlúčenie výsledných ohraničených rámčekov [13].

3.1.2 Fast R-CNN

V roku 2015 sa Ross Girshick, rovnaký autor ako v prípade R-CNN, snažil vyriešiť problémy predchádzajúcej metódy, aby vytvoril rýchlejší algoritmus detekcie objektov, ktorý nazval Fast R-CNN. Namiesto toho aby sa do prvej konvolučnej siete privádzali navrhované regióny, algoritmus privádza na konvolučnú vrstvu vstupný obraz. Výstup z konvolučnej vrstvy je mapa vlastností. Z mapy vlastností sa vypočítajú navrhované regióny, ktoré sa upravia na štvorcový tvar a použije sa RoI (Region of interest) združovanie. Vektor na výstupe z RoI združovania sa vloží ako vstup do plne prepojenej neurónovej siete. Z RoI združovania taktiež použijeme softmax vrstvu na predikovanie triedy objektu a posun pri ohraničovaní rámčekov [14].

Z tejto architektúry vyplýva, že Fast R-CNN je rýchlejší ako R-CNN pretože nie je nutné posielať všetkých 2000 navrhovaných regiónov do konvolučnej neurónovej siete. Pri trénovaní sa trénuje naraz CNN, klasifikátor a regresný model na zlúčenie ohraničených rámčekov. SVM klasifikátor bol nahradený softmax vrstvou, ktorá navyše sa používa aj ako regresný model na získanie súradníc pre ohraničené rámčeky [14].

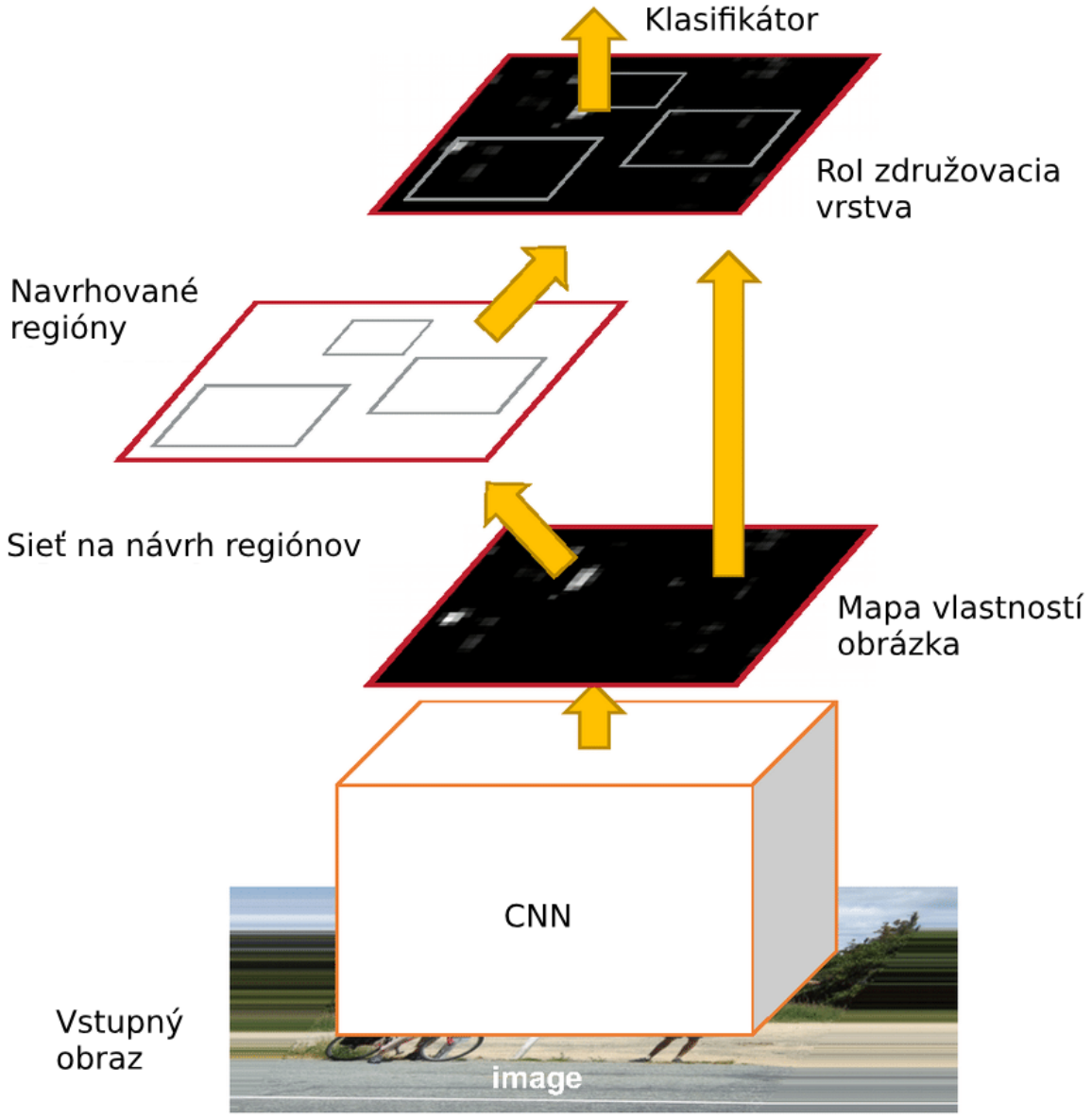


Obr. 16. Architektúra Fast R-CNN algoritmu [14].

3.1.3 Faster R-CNN

Neskôr v roku 2015 sa Ross Girshick a ostatní členovia z jeho tímu zo spoločnosti Microsoft Research snažili ešte vylepšiť algoritmus Fast R-CNN. Pri predchádzajúcich dvoch metódach sa na generovanie navrhovaných regiónov používa algoritmus selektívneho hľadania. Práve tento spôsob generovania navrhovaných regiónov spomaľoval celú metódu a tak sa vedci snažili tento spôsob vylepšiť. Novú vylepšenú metódu nazvali Faster R-CNN [15].

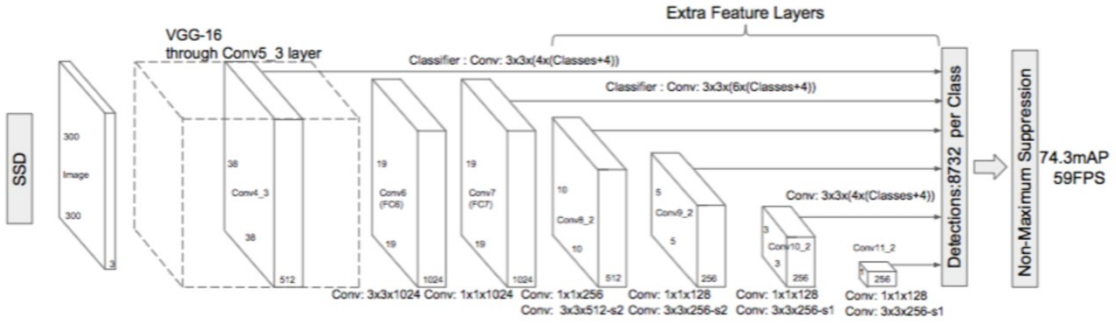
Ako aj pri Fast R-CNN architektúre sa vstupný obraz posiela do konvolučnej siete z ktorej výstup je konvolučná mapa vlastností. Namiesto použitia algoritmu selektívneho hľadania sa na generovanie navrhovaných regiónov používa samostatne plne prepojená neurónová sieť. Navrhované regióny sa potom použijú ako vstup do RoI združovacej vrstvy. Výstup RoI vrstvy je klasifikácia objektov pre navrhovaný región a hodnoty posunu pre ohraničujúce rámčeky. Vďaka týmto vylepšeniam je možné použiť Faster R-CNN metódu aj na detekovanie objektov v reálnom čase [15].



Obr. 17. Architektúra Faster R-CNN metódy [15].

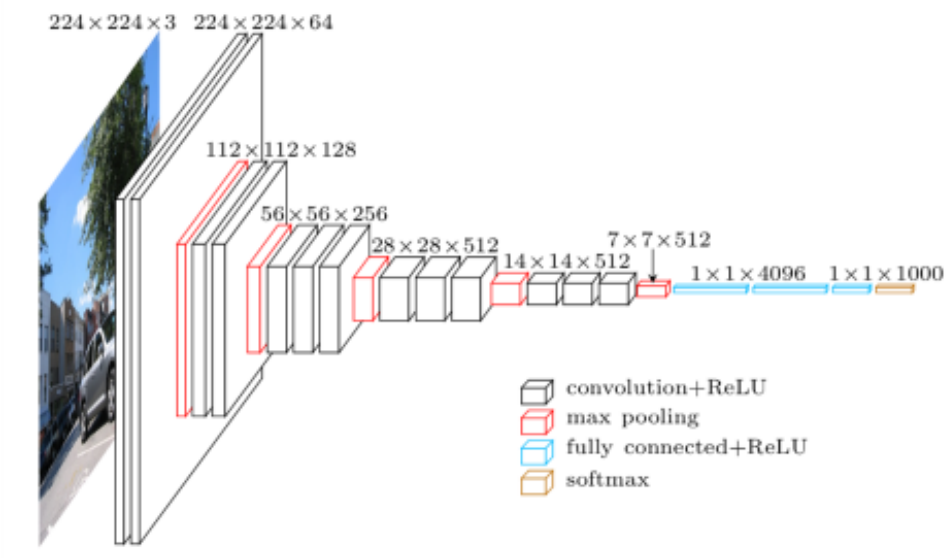
3.1.4 SSD (Single Shot Detector)

V roku 2016 vydal C. Szegedy a jeho tím novú metódu na detekovanie objektov. Okamžite zaznamenal úspech z hľadiska výkonu a presnosti pri úlohách na detekciu objektov. Dosiahli viac ako 74% mAP (priemerná presnosť) pri 59 snímkach za sekundu na štandardných dátových sadách ako je PascalVOC [32] alebo COCO [33]. Single Shot v preklade znamená „jediný vystreľ“ a v podstate popisuje, že lokalizácia a klasifikácia objektov sa robí jediným prechodom cez neurónovú sieť [16].



Obr. 18. Architektúra SSD metódy [16].

Ako je vidieť z návrhu tak SSD metóda je postavená na VGG-16 architektúre, ktorá sa skladá z dvanástich konvolučných vrstiev. Niektoré z týchto vrstiev sú nasledované maximálnymi združovacími vrstvami (maximum pooling layers). Následne nasledujú 4 plne prepojené vrstvy a na konci sa nachádza 1000 pásmový softmax klasifikátor. Táto architektúra je veľmi populárna pri vysoko kvalitných obrázkoch [16].



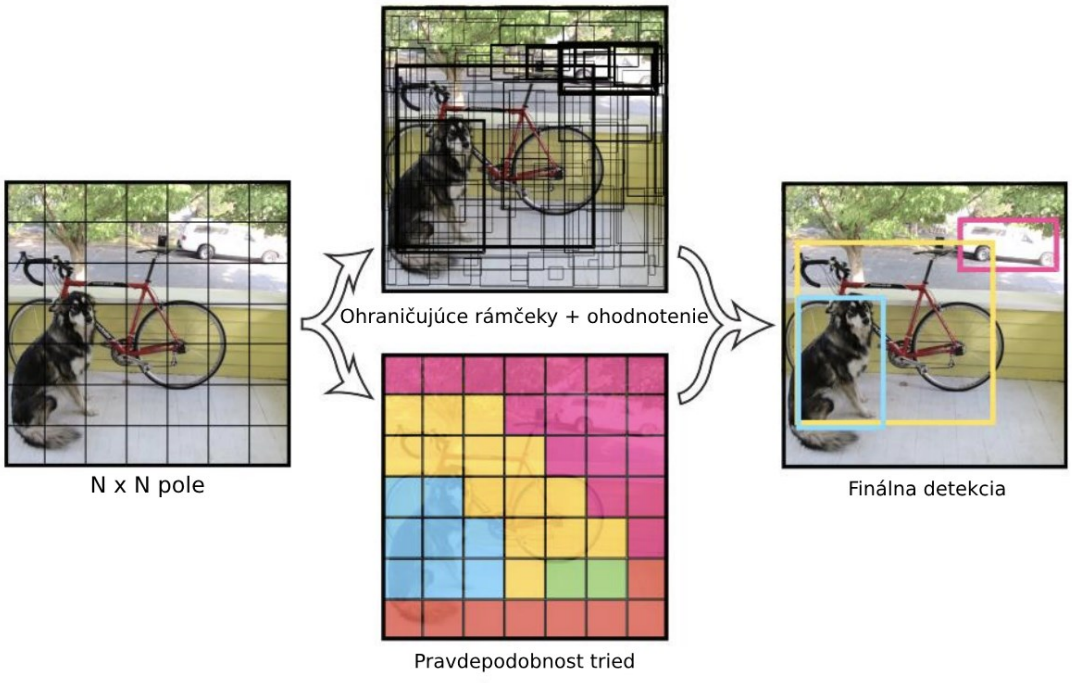
Obr. 19. VGG architektúra [16].

Na zlepšenie presnosti výsledkov sa používajú malé konvolučné filtre na predpovedanie tried objektov a posunov pri ohraničených rámčekov. Ďalej sa používajú aj samostatné filtre aby zvládli rozdiel v pomeroch strán pri získaní predvolených rámčekov. SSD na rozdiel od R-CCN metód dokáže lepšie detekovať objekty pri zníženej kvalite vstupného obrázka. Preto pri menších obrázkoch a detekovaní v reálnom čase prekonáva najmodernejší Faster R-CNN [16].

3.1.5 YOLO

YOLO metóda používa podobný prístup ako SSD metóda ale s tým rozdielom že neurónová sieť pracuje s celým obrázkom a nie len s navrhovanými regiónmi. Celú architektúru tvorí jedná neurónová sieť, ktorá v rámci jednej iterácie klasifikuje triedy objektov a zároveň predikuje ohraničujúce rámčeky. Vďaka tejto jednoduchej postupnosti spracovania, dokáže v súčasnej verzie predikovať 45 snímkov za sekundu na grafickej karte NVIDIA Titan X. Skratka YOLO vychádza z názvu You Only Look Once, čo v preklade znamená „Pozrieš sa iba raz“ [17].

Algoritmus YOLO pri detekcii rozdelí vstupný obraz do mriežky s rozmerom N x N. Podľa stredobodu jednotlivých objektov sa rozhoduje o tom, že do ktorej bunky patria objekty. V rámci týchto buniek sa vytvára M ohraničujúcich rámikov a určí sa pravdepodobnosť toho, že sa v nich nachádza daný objekt. Ak je pravdepodobnosť vyššia ako minimálna hodnota bunka sa použije. Pri jednotlivých ohraničujúcich rámčekoch sa predikuje ich stredobod v rámci mriežky a taktiež ich rozmery. Algoritmus sa riadi pravidlom, že v jednej bunke sa detekuje iba jeden objekt. Ako detekovaný objekt je najčastejšie sa vyskytujúca trieda z príslušnej bunky. Zvolené bunky sa ďalej presúvajú do neurónovej sieti, ktorej štruktúra je štandardná konvolučná sieť, obsahujúca zlučovaciu vrstvu nasledovanú dvomi plne prepojenými vrstvami [17].



Obr. 20. Znázornenie fungovania YOLO metódy [17].

3.1.6 Porovnanie metód

Každá metóda je prispôsobená na niečo iné a preto by mala byť určená podľa zámeru. Niektoré metódy sú pomalšie ale vyhodnocujú detekciu alebo klasifikáciu presnejšie ako rýchlejšie metódy. Na porovnanie metód sa používa mAP skóre (Mean Average Precision). Pri výpočte detekcie objektov sa vypočíta priemerná presnosť pre každú triedu z údajov na základe predpovedí modelu. Priemerná presnosť sa vzťahuje na plochu pod krivkou presnosti pre danú triedu. V poslednom kroku sa spraví priemer týchto priemerných presností pre danú triedu a výstupom je mAP skóre pre danú metódu alebo model. Nasledujúca tabuľka znázorňuje mAP skóre pre vybrané metódy s ich naučeným modelom spustené na rôznych datasetoch. Namerané hodnoty sú z oficiálnych publikácii od autorov metód [27].

Tab. 1. Porovnanie metód detekcie objektov [13,14,15,16,17]

Model / Dataset	PASCAL VOC 2007	PASCAL VOC 2010	PASCAL VOC 2012	COCO 2015
R-CNN	neuvedené	62,4%	neuvedené	neuvedené
Fast R-CNN	70,0%	68,8%	68,4%	neuvedené
Faster R-CNN	78,8%	neuvedené	75,9%	neuvedené
YOLO	63,7%	neuvedené	57,9%	neuvedené
YOLOv2	78,6%	neuvedené	neuvedené	44,0%
SSD	83,2%	neuvedené	82,2%	48,5%

3.2 Frameworky hlbokého učenia

Vytvoriť a implementovať neurónovú sieť od úplného základu je veľmi náročne a môže to trvať niekoľko mesiacov alebo rokov aby boli dosiahnuté aspoň porovnateľné výsledky s dnešnými frameworkami hlbokého učenia. Preto sa v dnešnej dobe mnoho vývojárov prikláňa ku optimalizovaným a komplexným frameworkom, ktoré už majú mnoho vecí a vlastnosti implementované.

Framework hlbokého učenia je rozhranie, knižnica alebo nástroj, ktorý umožňuje ľahšie a rýchlejšie vytvárať modely hlbokého učenia bez toho, aby používateľ musel riešiť do detailov základné algoritmy. Poskytujú jasný a výstižný spôsob definovania modelov pomocou kolekcie prednastavených a optimalizovaných komponentov. Medzi najznámejšie frameworky hlbokého učenia patrí:

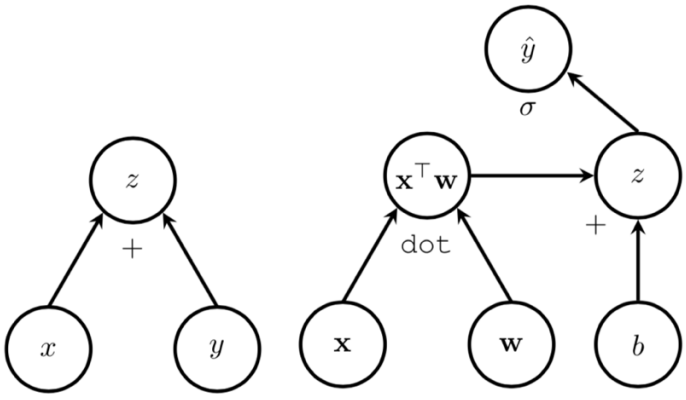
- TensorFlow,
- Darknet,
- PyTorch,
- Caffe,
- Keras.

3.2.1 TensorFlow

TensorFlow je open source knižnica určená na numerické výpočty a rozsiahle strojové učenie. Na reprezentáciu výpočtov sa používajú grafy dátových tokov. Uzly v grafe reprezentujú matematické operácie a hrany reprezentujú viacrozmerné pole dát (tenzory). To vedie k nízkoúrovňovému programovaciemu modelu, v ktorom sa najprv definuje graf dátových tokov a potom sa vytvorí relácia na spustenie častí grafu cez množinu lokálnych a vzdialených zariadení. Flexibilná architektúra podporuje prácu na CPU a GPU a taktiež podporuje pracovať s jedným alebo viacerými jadrami. Framework je naprogramovaný v programovacích jazykoch Python, C++ a CUDA a je možné s ním pracovať na platformách ako je Windows, Linux, MacOS, Android, iOS. Pôvodne Tensorflow bol vyvinutý členmi tímu Google Brain pre interné použitie vo firme Google. Od roku 2015 bol zverejnený pod Apache 2.0 open-source licenciou [18].

3.2.1.1 Architektúra

V TensorFlow sú zastúpene algoritmy ako výpočtové grafy. Výpočtový alebo dátový graf sa skladá s uzlov, ktoré predstavujú operácie. Hrany medzi uzlami predstavujú dáta, ktoré prúdia medzi operáciami. Ak výstupná premenná *z* výsledkom binárnej operácie medzi vstupmi *x* a y, potom sa vytvorí orientovaná hrana zo vstupu *x* a *y* do *z* [18].



Obr. 21. Príklady výpočtových grafov [18].

Hlavnou výhodou reprezentovania algoritmu vo forme grafe je nielen vizuálne vyjadrenie závislosti ale aj všeobecné vyjadrenie závislostí medzi jednotkami modelu. Operácia môže predstavovať matematickú rovnicu, premennú, konštantu, smernicu pre kontrolu toku alebo operácie pri práci so súbormi. Konštanta môže byť predstavovaná za operáciu, ktorá nemá žiadne vstupy a na výstupe je vždy rovnaká hodnota. Analogicky, premenná je len operácia, ktorá nemá žiadny vstup a produkuje aktuálny stav alebo hodnotu danej premennej. Každá operácia musí mať svoju implementáciu [18].

Hrany medzi uzlami reprezentujú tok dát z jednej operácie do druhej. V TensorFlow sa označujú ako tenzory. Tenzor je viacrozmerné pole homogénnych hodnôt s pevne stanoveným typom. Jeho tvar je popísaný ako n-tica, ktorá popisuje počet komponentov v každom rozmere. V matematickom smere je tenzor všeobecne zapísaná dvojrozmerná matica, jednorozmerný vektor alebo skalár, ktorý ma nulovú hodnosť. Tenzor sám nedrží ani neukladá hodnoty do pamäte. Jeho funkcia je poskytnúť rozhrania na načítanie hodnoty na ktorú odkazuje. Pri jednoduchej operácii ako je $x + y$ sa vráti objekt tenzora, ktorý sa následne môže použiť ako vstup do iných výpočtov [18].

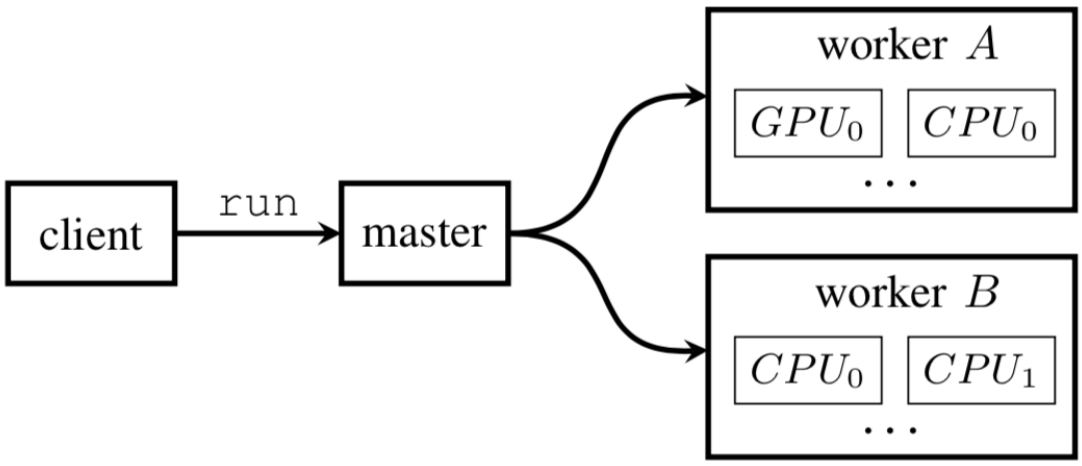
Pri rôznych situáciách je ale nutnosť model grafu vykonávať od začiatku do konca viackrát pre jeden experiment. Každá iterácia experimentu zničí väčšinu tenzorov. Často je ale potrebné udržiavať stav a hodnoty počas iterácii napríklad váhy a parametre pri neurónových sieťach. Na tento účel sú v TensorFlow architektúre operácie nazvané premenné. Premenné možno opísať ako perzistentné, premenlivé operácie, ktoré držia svoju hodnotu počas iterovania modelu grafu [18].

Vykonávanie operácii a vyhodnocovanie tenzorov sa môže vykonávať len v špeciálnom prostredí, ktoré sa nazýva relácia(session). Jednou z jej úloh je alokovať a spravovať zdroje ako napríklad vyrovnávaciu pamäť premenných. Interface relácie taktiež poskytuje aj bežiacu rutinu, ktorá je primárnym vstupným bodom pre vykonávanie častí alebo celého výpočtového grafu. Táto metóda berie uzly grafy ako vstup, ktorých tenzory by mali byť vypočítane a vrátene na výstupe. TensorFlow ponúka aj možnosť ľubovoľne namapovať uzly pre spustenie. Po spustení behu začne na požadovaných výstupných uzloch a smerom dozadu skúma závislosti grafov a počíta úplné tranzitívne uzavretie všetkých uzlov, ktoré musia byť spustené. Tieto uzly môžu byť priradené k jednej alebo viacerým vykonávajúcim jednotkám CPU alebo GPU na jednom alebo viacerých počítačoch. O synchronizáciu a spúšťanie uzlov na viacerých jednotkách sa stará algoritmus umiestnenia (TensorFlow's placement algorithm) [18].

3.2.1.2 Vykonanie modelu

Na vykonávanie výpočtových grafov, ktoré sa skladajú z rôznych prvkov, TensorFlow rozdeľuje úlohy do štyroch skupín: master, klient (client), skupina pracovníkov (worker) a viacerých zariadení (devices). Keď klient požaduje vyhodnotenie grafu pomocou spúšťacej rutiny, táto požiadavka sa pošle master procesu. Master následne deleguje úlohy jednému alebo viacerým pracovníkom a koordinuje ich vykonávanie. Každý pracovník je následne zodpovedný za správu nad jedným alebo viacerými zariadeniami. Medzi zariadenia patria fyzicky spracovateľské jednotky, pre ktoré sú implementované jadrá operácie [18].

V rámci tohto modelu existujú dve možnosti škálovania. Prvý stupeň škálovania sa týka počtu strojov na ktorých sa graf vykonáva. Druhý stupeň sa týka počtu zariadení na daných strojoch. Na jednom stroji môžu byť napríklad tri nezávislé GPU a dve CPU jadrá [18].



Obr. 22. Vizualizácia vykonávania modelu s viacerými strojmi [18].

Zariadenia sú najmenšími a najzakladanejšími entitami v realizačnom TensorFlow modeli. Všetky uzly v grafe musia byť namapované na dostupné zariadenia, ktoré sa majú vykonať. V praxi je zariadenie najčastejšie CPU alebo GPU. Tensorflow ale podporuje aj ďalšie druhy spracovateľských jednotiek. V roku 2016 spoločnosť Google predstavila svoju vlastnú spracovateľskú jednotku Tensor Processing Unit (TPU), ktorá má aplikačne špecificky integrovaný obvod, optimalizovaný špeciálne pre rýchle výpočty tenzorov [18].

3.2.2 Darknet

Darknet je open source framework na prácu s neurónovými sieťami. Framework je napísaný v programovacích jazykoch C a CUDA a vyznačuje sa najmä rýchlosťou a jednoduchou inštaláciou. Pri výpočtoch je možné použiť výpočtový výkon z CPU alebo GPU. Darknet je

nainštalovaný len s dvomi voliteľnými závislosťami. Prvá je OpenCV knižnica, ktorá poskytuje možnosti pri spracovaní obrazu. Druhá je CUDA podpora na použitie výpočtovej sily z GPU [19].

Framework má priamu podporu metódy na detekciu objektov YOLO. Na grafickej karte, NVIDIA Titan X, spracuje 40-90 obrázkov za sekundu pri úspešnosti detekcie 78.6% na datasete VOC 2007. Darknet taktiež poskytuje možnosť spúšťania rekurentných neurónových sietí. Rekurentné neurónové siete sú výkonnými modelmi pre reprezentáciu dát, ktoré sa menia v čase. Darknet tieto siete dokáže spustiť aj bez podpory CUDA alebo OpenCV. Framework obsahuje aj neurónovú sieť, ktorá predpovedá najpravdepodobnejšie najbližšie ťahy v hre GO [19].

3.2.2.1 Inštalácia

Inštalácia a implementácia Darknet frameworku je veľmi jednoduchá. Celá inštalácia spočíva v troch riadkoch. Prvý riadok stiahne Darknet framework z oficiálneho repozitára vývojára do súčasného adresára. Druhý riadok otvorí naklonovaný framework. Následne sa spusti inštaláciou pomocou tretieho príkazu. Adresár repozitára obsahuje *MakeFile* súbor, ktorý sa postará o celú inštaláciu [19].

```
1.      git clone https://github.com/pjreddie/darknet.git
2.      cd darknet
3.      make
```

Predvolená inštalácia nastaví použitie iba výpočtovej sily z CPU. V prípade ak je na danom stroji nainštalovaný program na podporu CUDA, je možné nastaviť aby framework pracoval s GPU. Darknet na GPU je 500-krát rýchlejší ako na CPU. Celé nastavovanie predvoleného zariadenia sa nastaví v súbore *MakeFile*. Po zmene hodnoty parametra GPU z 0 na 1, je nutné znova prekompilovať framework pomocou príkazu *make.*

```
1.      GPU=1
2.      make
```

V tomto momente je Darknet framework pripravený na použitie.

3.2.2.2 Pretrénované modely

Darknet je veľmi vhodný na testovanie a experimentovanie s pretrénovanými modelmi. Môžem ho napríklad použiť na klasifikovanie obrázkov pre ImageNet Challenge, ktorý obsahuje 1000 tried objektov. Pred spustením klasifikácie obrázkov je nutné stiahnuť súbor s klasifikačnými váhami pre daný model [19].

```
1.      wget https://pjreddie.com/media/files/darknet19.weights
```

V danom prípade som použil pretrénovaný model z neurónovej siete darknet19, ktorá obsahuje 19 konvolučných vrstiev. Pre každý model sú dôležité dva súbory. Prvý je súbor s váhami a druhý je konfiguračný súbor. Konfiguračný súbor pre model darknet19 a dátový súbor pre ImageNet sú súčasťou frameworku. Na spustenie klasifikácie obrázku (data/dog.jpg) sa používa nasledovný príkaz [19].

```
1.      ./darknet classifier predict cfg/imagenet1k.data cfg/darknet19.cfg darknet19.weights data/dog.jpg
```

Obr. 23. Testovací obrázok použitý pri klasifikácii [19].

Na výstupe z klasifikácie pomocou Darknet frameworku je možné vidieť aké vrstvy boli použité pri klasifikácii a v dolnej časti sa nachádza 5 najlepších tried pre daný obrázok.

```
layer     filters    size              input                output
    0 conv     32  3 x 3 / 1   256 x 256 x   3   ->   256 x 256 x  32  0.113 BFLOPs
    1 max          2 x 2 / 2   256 x 256 x  32   ->   128 x 128 x  32
    2 conv     64  3 x 3 / 1   128 x 128 x  32   ->   128 x 128 x  64  0.604 BFLOPs
    3 max          2 x 2 / 2   128 x 128 x  64   ->    64 x  64 x  64
    4 conv    128  3 x 3 / 1    64 x  64 x  64   ->    64 x  64 x 128  0.604 BFLOPs
    5 conv     64  1 x 1 / 1    64 x  64 x 128   ->    64 x  64 x  64  0.067 BFLOPs
    6 conv    128  3 x 3 / 1    64 x  64 x  64   ->    64 x  64 x 128  0.604 BFLOPs
    7 max          2 x 2 / 2    64 x  64 x 128   ->    32 x  32 x 128
    8 conv    256  3 x 3 / 1    32 x  32 x 128   ->    32 x  32 x 256  0.604 BFLOPs
    9 conv    128  1 x 1 / 1    32 x  32 x 256   ->    32 x  32 x 128  0.067 BFLOPs
   10 conv    256  3 x 3 / 1    32 x  32 x 128   ->    32 x  32 x 256  0.604 BFLOPs
   11 max          2 x 2 / 2    32 x  32 x 256   ->    16 x  16 x 256
   12 conv    512  3 x 3 / 1    16 x  16 x 256   ->    16 x  16 x 512  0.604 BFLOPs
   13 conv    256  1 x 1 / 1    16 x  16 x 512   ->    16 x  16 x 256  0.067 BFLOPs
   14 conv    512  3 x 3 / 1    16 x  16 x 256   ->    16 x  16 x 512  0.604 BFLOPs
   15 conv    256  1 x 1 / 1    16 x  16 x 512   ->    16 x  16 x 256  0.067 BFLOPs
   16 conv    512  3 x 3 / 1    16 x  16 x 256   ->    16 x  16 x 512  0.604 BFLOPs
   17 max          2 x 2 / 2    16 x  16 x 512   ->     8 x   8 x 512
   18 conv   1024  3 x 3 / 1     8 x   8 x 512   ->     8 x   8 x1024  0.604 BFLOPs
   19 conv    512  1 x 1 / 1     8 x   8 x1024   ->     8 x   8 x 512  0.067 BFLOPs
   20 conv   1024  3 x 3 / 1     8 x   8 x 512   ->     8 x   8 x1024  0.604 BFLOPs
   21 conv    512  1 x 1 / 1     8 x   8 x1024   ->     8 x   8 x 512  0.067 BFLOPs
   22 conv   1024  3 x 3 / 1     8 x   8 x 512   ->     8 x   8 x1024  0.604 BFLOPs
   23 conv   1000  1 x 1 / 1     8 x   8 x1024   ->     8 x   8 x1000  0.131 BFLOPs
   24 avg                       8 x   8 x1000   ->  1000
   25 softmax                                       1000
Loading weights from darknet19.weights...Done!
data/dog.jpg: Predicted in 0.846738 seconds.
42.55%: malamute
22.93%: Eskimo dog
12.51%: Siberian husky
 2.76%: bicycle-built-for-two
 1.20%: mountain bike
```

Obr. 24. Výstup z klasifikácie pomocou Darknet frameworku.

3.2.2.3 Trénovanie

Ako aj pri ostatných úlohách hlbokého učenia, najdôležitejšia časť učenia je príprava datasetu. Veľmi dobrý dataset ponúka spoločnosť Google. *OpenImagesV4* je veľký dataset, ktorý obsahuje až 600 rozličných tried objektov. Dataset obsahuje aj anotácie ohraničeného rámika pre tieto objekty. Pre detekovanie viacerých tried objektov je potreba mať čím najviac príkladov pre danú triedu.

Každý tréning strojového učenia zahŕňa najprv rozdelenie dataset náhodne do dvoch setov. Tréningový set obsahuje dáta na ktorých trénujeme model. Zvyčajne to býva medzi 70% až 90% dát z celého vybraného datasetu. Dáta v testovacom sete slúžia následne na testovanie natrénovaného modelu. Typicky je to 10% až 30% z celkového súboru. Dôležité aby sa jedna snímka nenachádzala v obidvoch setoch súčasne.

Pri trénovaní vlastného detektoru objektov je dobré využiť existujúce modely, ktoré sú natrénované na veľkých datasetoch aj v prípade ak nový detekovaný objekt nebol súčasťou toho datasetu. Tento proces sa nazýva transfer learning. Pomocou pretrénovaného modelu sa použijú natrénované váhy čo zrýchľuje učenie nové modelu.

Na trénovanie vlastného modelu pomocou Darknet frameworku je nutné vytvoriť dátový súbor, napríklad darknet.data, ktorý obsahuje informácie a špecifikáciu pre detektor objektov.

```
1.      classes = 1
2.      train = object_train.txt
3.      valid = object_valid.txt
4.      names = classes.names
5.      backup = /
```

V tomto prípade sa trénuje model, ktorý je schopný detekovať len jednu triedu. Textové súbory *object_train.txt* a *object_valid.txt* obsahujú cesty k obrázkom na trénovanie a testovanie. Súbor *classes.names* obsahuje len názov novej triedy. Darknet pri trénovaní odporúča použiť pri trénovaní vlastného modelu na detekciu objektov konfiguračný súbor z YOLOv3 modelu, ktorý je súčasťou frameworku a nazýva sa *yolov3.cfg*. Ako bolo vyššie uvedené nie je vhodné začínať trénovanie úplne od nuly, ale je lepšie využiť váhy z pretrénovalo modelu, ktoré sú uložené v súbore *pre-trained.weights.* Následne sa použije jednoduchý príkaz na spustenie trénovania [19].

```
1.      ./darknet detector train darknet.data darknet-yolov3.cfg pre-trained.weights
```

Trénovanie vlastného modelu je veľmi náročný proces na výpočtovú silu. Preto sa odporúča trénovať modely na GPU. Po úspešnom natrénovaní sa nový model nachádza v adresári *backup*, ktorý sa nastavuje v súbore *darknet.data*. Čas a kvalita trénovania závisí od rôznych parametrov, ktoré sa dajú nastavovať v konfiguračnom súbore. Medzi parametre napríklad patrí parameter *batch*, ktorý je na začiatku nastavený na hodnotu 64. V tomto prípade to znamená, že 64 obrázkov sa použije v jednej iterácii na aktualizovanie parametrov neurónovej siete. Na trénovania má vplyv aj nastavenia rozlíšenia vstupného obrázka, čím väčší obrázok tým model bude presnejší ale trénovanie trvá podstatne dlhšie. Prednastavené rozlíšenie je 416 x 416. Nakoniec je dôležité určiť, koľko iterácii by mal tréningový proces bežať pomocou parametru *max_batches*. Zvyčajne sa používa na počet iterácii pre objektový detektor n-tried nasledujúci matematický vzorec(12) [19].

$$\text{max\_batches} = 2000 * n \tag{12}$$

4 HLBOKÉ NEURÓNOVE SIETE V OPENCV

OpenCV je najznámejšia knižnica na manipuláciu s obrazom. Jej zameranie je prevažne na počítačové videnia a spracovanie obrazu. Za 19 rokov si knižnica získala obrovské množstvo používateľov a stala sa štandardom v oblasti počítačového videnia. Medzi hlavné výhody knižnice patrí ponuka prostredia pre programovacie jazyky C++, Python, Matlab, C# a Java. Taktiež podporuje najpoužívanejšie operačné systémy Windows, Linux, MacOS, iOS a Android. Naprogramovaná je v C/C++ preto knižnica podporuje prácu na viacerých jadrách procesora alebo grafickej karty. OpenCV knižnica sa neustále vyvíja a pridávajú sa nové a zaujímavé moduly. Od verzie 3.1 je dostupný nový modul, DNN modul, na podporu hlbokých neurónových sietí [20].

V posledných rokoch sa v mnohých oblastiach ukazuje, že výsledky pomocou hlbokého učenia prevyšujú výsledky klasických algoritmov. To platí a pre oblasť počítačového videnia, kde sa množstvo problémov rieši pomocou neurónových sietí. Vzhľadom na túto skutočnosť bola nutnosť pridať podporu neurónových sieti do OpenCV knižnice. Modul na spúšťanie hlbokých sietí výrazne zjednodušuje postup vytvárania algoritmov, ktoré kombinujú rýchlosť klasického počítačového videnia a pozoruhodnú generalizačnú schopnosť hlbokých neurónových sietí [20].

Hlavnou výhodou dnn modulu je načítavanie a spustenie neurónových sietí. V tomto prípade môže byť model vytvorený z niektorých podporovaných frameworkov pre hlboké učenie. Medzi podporované frameworky patrí:

- Caffe,
- TensorFlow,
- Torch,
- Darknet,
- Modely v ONNX formáte [21].

ONNX je otvorený formát reprezentujúci modely hlbokého učenia. S ONNX môžu vývojári jednoduchšie presúvať modely medzi najmodernejšími nástrojmi a zvoliť si kombináciu, ktorá je pre nich najvhodnejšia [21].

Podpora piatich populárnych frameworkov poskytuje možnosť jednoducho kombinovať výsledky modelov. Pri načítaní modelu sa každý model prevedie na internú reprezentáciu, ktorá je blízka reprezentácii použitej z frameworku Caffe. Je to z historických dôvodov, pretože

prvý podporovaný framework bol práve Caffe. Napriek tomu medzi reprezentáciami neexistuje žiadna vzájomná korešpondencia [21].

V najnovšej verzii DNN modul podporuje až 32 vrstiev. Od tých základných ako konvolučné a plne prepojené až po špecializované ako napríklad AbsVal, Crop, MaxPooling atď. Okrem podpory jednoduchých vrstiev modul podporuje aj špecifické architektúry neurónových sietí. Modul obsahuje príklady pre klasifikáciu (AlexNet, GoogLeNet, ResNet, SqueezeNet), segmentáciu (FCN, ENET), detekciu objektov (SSD) [21].

Ako aj pri ostatných moduloch pre OpenCV, tak aj DNN modul poskytuje veľmi dobrú dokumentáciu s praktickými príkladmi. Príklady sú pripravené v programovacom jazyku C++ a Python. Dokumentácia sa nachádza na nasledujúcom odkaze https://docs.opencv.org/4.1.0/d6/d0f/group__dnn.html [21].

4.1 Presnosť

Vzhľadom k tomu, že DNN modul načítava a prevádza modely z rôznych rámcov do svojej vnútornej reprezentácie, vznikli rôzne diskusie či po prevedení modelu ostala zachovaná kvalita natrénovaného modelu. Na porovnanie boli testované modely z existujúcich príkladov pre rôzne frameworky na klasifikačných problémoch. Merania boli vykonané na validačnom datasete ImageNet 2012, ktorý obsahuje 50000 položiek. Namerané hodnoty boli porovnávane s oficiálnymi výsledkami daných modelov [22].

Tab. 2. Výsledky hodnotenia kvality pre klasifikačnú úlohu [22].

Model	Oficiálna zverejnená hodnota	Nameraná hodnota v DNN	Priemerný rozdiel medzi elementmi	Maximálny rozdiel výstupu
AlexNet (Caffe)	80,2%	79,1%	$6{,}5 \times 10^{-10}$	$3{,}01 \times 10^{-6}$
GoogLeNet (Caffe)	88,9%	88,5%	$1{,}18 \times 10^{-9}$	$1{,}33 \times 10^{-5}$
GoogLeNet (Tensorflow)	nezverejnené	89,4%	$1{,}84 \times 10^{-9}$	$1{,}47 \times 10^{-5}$
ResNet – 50 (Caffe)	92,2%	91,8%	$8{,}73 \times 10^{-10}$	$4{,}29 \times 10^{-6}$
SqueezeNet v1.1 (Caffe)	80,3%	80,4%	$1{,}91 \times 10^{-9}$	$6{,}77 \times 10^{-6}$

Z tabuľky vyplýva, že pre klasifikačné problémy je rozdiel presnosti veľmi minimálny medzi pôvodným frameworkom a použitím DNN modulu v OpenCV. Preto môžem tvrdiť, že modul sa môže používať pri daných modelov bez strachu z nepredvídateľných výsledkov. Drobný rozdiel medzi publikovanými a experimentálne získanými výsledkami je možné odôvodniť tým, že autori modelov použili výkonnostný výkon GPU a pri tomto porovnaní boli výsledky získané len pomocou CPU. Ďalším problémom môže byť spracovanie vstupného obrázku, kde každá knižnica dekóduje formát jpeg inak.

4.2 Produktivita

Jedným z cieľov, ktorý si vývojári z OpenCV pri vývoji DNN modulu stanovili, bol dosiahnuť slušný výkon na rôznych architektúrach pri použití výpočtovej sily na CPU bez použitia grafickej karty. Experimenty sa vykonávali s použitím Intel Core i7-6700k [22].

Tab. 3. Porovnanie produktivity pôvodných frameworkov a DNN modulu [22].

Model	Rozlíšenie obrázka	Výkon pôvodného frameworku na CPU	Výkon DNN na CPU	Rozdiel
AlexNet (Caffe)	227x227	23,7 ms	14,7 ms	1,6x
GoogLeNet (Caffe)	224x224	44,6 ms	20,1 ms	2,2x
GoogLeNet (Tensorflow)	224x224	17,9 ms	21,1 ms	0,8x
ResNet – 50 (Caffe)	224x224	70,2 ms	58,8 ms	1,2x
SqueezeNet v1.1 (Caffe)	227x227	12,4 ms	5,3 ms	2,3x

Z tabuľky výkonnosti vyplýva, že vo väčšine prípadoch DNN modul presahuje výkonnosť pôvodných frameworkov.

4.3 Ďalšie plány DNN modulu

Modul si získal mnoho fanúšikov medzi používateľmi práve kvôli rýchlosti vykonávania modelov a jednoduchej implementácie. Preto vývojári modulu už teraz pracujú na vylepšeniach. Jedným z navrhovaných vylepšení je pridanie vizualizácie siete ako aj rozšírenie sady

podporovaných vrstiev a frameworkov. Pokiaľ ide o výkon, tak momentálne sa experimentuje so znížením bitness výpočtu, ktorý sa nazýva kvantovanie. Tento krok by mal viesť k zvýšeniu rýchlosti ale možno na úkor presnosti. Ďalšou optimalizačnou myšlienkou je paralelné vykonávanie vrstiev. V súčasnej implementácii pracuje v jeden čas len jedna vrstva. V niektorých prípadoch však môže byť výpočet paralelizovaný na úrovni samostatných vrstiev. To by mohlo rozdeliť prácu do viac vlákien, čím sa znížia náklady na výpočet [22].

II. PRAKTICKÁ ČASŤ

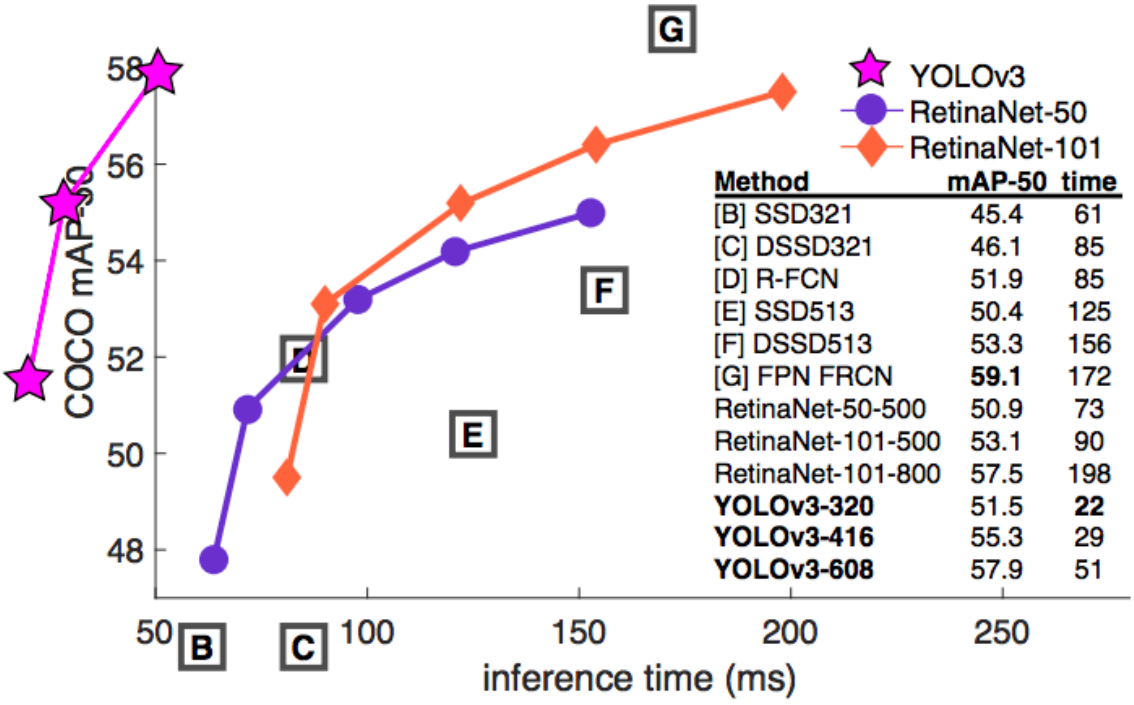
5 METÓDA CHYTRÉHO ORÉZÁVANIA OBRÁZKOV

V praktickej časti diplomovej práce navrhnem, naprogramujem a implementujem novú metódu orezávania obrázkov v mediálnej knižnici DAM. Najdôležitejším prvkom novej metódy je použitie umelej inteligencie na detekciu objektov, ktorá by mala pomôcť pri určovaní takzvaného ohniskového bodu na obrázkoch. Podľa ohniskového bodu by sa mali následne generovať náhľady pre rôzne rozlíšenia. Celé riešenie by som rozdelil do troch kategórii, ktoré postupne opíšem. Medzi tri hlavné kategórii patrí detekcia objektov, webové riešenie na úpravu ohniskového bodu a prepojenie medzi detekovanými objektami a webovým riešením.

5.1 Detekcia objektov a výber ohniskového bodu

Najdôležitejšia časť celého riešenia je implementácia algoritmu na detekciu objektov. Ako programovací jazyk pri implementácii algoritmu na detekciu objektov som si vybral Python v najnovšej verzii 3.7.2. Vybral som si ho kvôli predošlým skúsenostiam, jednoduchej syntaxi a veľkej podpore pri práci s umelou inteligenciou a neurónovými sieťami.

Počas výberu vhodného riešenia na detekciu objektov som bral do úvahy aj finančnú stránku výpočtovej sily. Mojím cieľom bolo poskytnúť rýchly algoritmus, ktorý je schopný pracovať aj na CPU bez použitia grafickej karty. Ako vhodné riešenie mi prišlo použiť knižnicu OpenCV, ktorá sa prevažne zameriava na počítačové videnie a spracovanie obrazu, ktorá ma priamo modul DNN na prácu s umelými neurónovými sieťami. Pri experimentovaní som mal možnosť si vyskúšať prácu s rôznymi frameworkami ako je Darknet, TensorFlow alebo Caffe. Veľkým pozitívom pri DNN module je priama podpora spomínaných frameworkov, čím odpadá nutnosť inštalovať rôzne knižnice pri experimentovaní. Rozhodoval som sa medzi najznámejším frameworkom TensorFlow a celkom novým a menej známym frameworkom Darknet. Nakoniec som si vybral framework Darknet, kvôli rýchlosti detekcie, jednoduchej implementácii a naučeného modelu na detekciu a rozoznávanie objektov z obrazu YOLOv3. YOLOv3 pri porovnaní s ostatnými súčasnými modelmi vyšiel najlepšie viď obr. 25 [23].



Obr. 25. Porovnanie YOLOv3 s ostatnými modelmi [23].

5.1.1 Naučený model YOLOv3

YOLOv3 model je jedným z najpopulárnejších objektových detektorov v počítačovom videní. Nakonfiguroval, naučil a implementoval ho rovnaký ako autor ako v prípade frameworku Darknet. Z toho dôvodu je optimalizovaný a veľmi rýchly s použitím OpenCV a Darknet frameworku. Práve pri použití kombinácie Darknet a OpenCV s DNN modulom je možné dosiahnuť až 9x rýchlejšie výsledok ako pri použití OpenMP API.

Je to najnovšia verzia populárnej metódy na detekciu objektov YOLO. Naučený model obsahuje 80 tried, ktoré je schopný detekovať. V mojom prípade je to dostačujúce množstvo. Obsahuje základné triedy ako napríklad osoba, auto, mačka, pes ale aj menej používané triedy ako je kniha, počítač, pizza a podobne. Pri testovaní a implementácii som pracoval s CPU na operačnom systéme macOS, preto bola pre mňa rýchlosť detekcie veľmi rozhodujúca. Z tabuľky porovnania vidím, že najlepší čas detekcie dostávam ak použijem grafickú kartu ale v mojom prípade je dostačujúca kombinácia Darknet + OpenCV na procesore 3 GHz Intel Core i5. Čas detekcie vyšiel približne 232 ms.

Tab. 4. Porovnanie YOLOv3 modelu pri použití rôznych frameworkov, výpočtovej sily a operačného systému [23].

Operačný systém	Framework	CPU/GPU	Čas detekcie
Linux 16.04	Darknet	12x Intel Core i7-6850K CPU @ 3.60GHz	9370 ms
Linux 16.04	Darknet + OpenMP	12x Intel Core i7-6850K CPU @ 3.60GHz	1942 ms
Linux 16.04	Darknet + OpenCV	12x Intel Core i7-6850K CPU @ 3.60GHz	220 ms
macOS	Darknet + OpenCV	2.5 GHz Intel Core i7 CPU	400 ms
Linux 16.04	Darknet	NVIDIA GeForce 1080 Ti GPU	23 ms
macOS	Darknet	2.5 GHz Intel Core i7 CPU	7260 ms
macOS	Darknet	3 GHz Intel Core i5 CPU	4300 ms
macOS	Darknet + OpenCV	3 GHz Intel Core i5 CPU	232 ms

5.1.2 Implementácia detektora objektov

Inštaláciou OpenCV knižnice a Pythonu sa budem venovať v neskoršej kapitole. Ešte pred implementáciu a použitím YOLOv3 modelu som musel stiahnuť do nového projektu konfiguračný súbor, súbor s naučenými váhami a textový súbor s triedami. Súbory sa nachádzajú v repozitári autora modelu. Použijem *wget* program, ktorý slúži na sťahovanie súborov pomocou HTTP/HTTPS alebo FTP protokolu. Daný program je predvolenou súčasťou operačných systémov macOS a Linux.

```
1.      wget https://pjreddie.com/media/files/yolov3.weights
2.      wget https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg?raw=true -
        O ./yolov3.cfg
3.      wget https://github.com/pjreddie/darknet/blob/master/data/coco.names?raw=true -O ./clas-
        ses.txt
```

V tomto kroku už mám všetko potrebné na implementáciu detektora. Začnem nastavením neurónovej siete.

```
1.      modelConfiguration = "src/yolov3/yolov3.cfg"
2.      modelWeights = "src/yolov3/yolov3.weights"
3.      net = cv.dnn.readNetFromDarknet(modelConfiguration, modelWeights)
4.      net.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
5.      net.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)
```

Stiahnuté súbory následne použijem pri konštruovaní neurónovej siete. V tomto prípade som použil framework Darknet, ktorý vyžaduje ako prvý parameter cestu ku konfiguračnému súboru a druhý parameter je cesta k súboru s váhami. Na riadku 4 a 5 je nastavenie prostredia, kde vyberám na čom bude neurónová sieť pracovať. V mojom prípade OpenCV a CPU.

Ďalším dôležitým krokom implementácie je pripraviť vstupný obraz pre nastavenú neurónovú sieť. Tu sa mi veľmi osvedčilo použiť základné funkcie knižnice OpenCV. Najprv si overím či vstupný obrázok sa nachádza v adresári. Ak nie tak vyhodím podmienku a ďalej nepokračujem.

```
1.      image = cv.imread(sourceImage)
2.      if image is None:
3.          raise ValueError('Missing source file ' + sourceImage)
```

Vstupný obraz následne musím spracovať aby bol vhodný ako zdroj do neurónovej siete. Neurónová sieť vyžaduje ako vstup obraz v binárnom tvare. Preto sa používa formát BLOB čo je skratka pre Binary Large OBject a označuje skupinu pripojených pixelov v binárnom obraze.

```
1.      imageBlob = cv.dnn.blobFromImage(image, 1 / 255, (416, 416), [0, 0, 0], 1, False)
2.      net.setInput(imageBlob)
```

Na prevod do binárneho formátu mi pomohla funkcia *blobFromImage*. Každý framework vyžaduje iné vstupné parametre danej funkcie. Parametre funkcie sú nasledovné

1. image – vstupný obraz,
2. scalefactor – miera zmenšenia vstupného obrazu,
3. size – veľkosť ktorú požaduje konvolučná sieť,
4. mean – stredná hodnota, ktorá sa odpočíta od každej farebnej zložky,
5. swapRB – OpenCV knižnica predpokladá, že farebné zložky sú vo formáte BGR, preto tento parameter slúži na otočenie zložiek na formát RGB.

Binárnu podobu obrázku následne vložíme ako vstup do neurónovej siete. Najdôležitejšia funkcia siete v OpenCV je *forward* funkcia. Keďže chcem aby vstup bežal cez celú sieť potrebujem vložiť ako parameter do funkcie výstupné vrstvy siete.

```
1.  layers = net.getLayerNames()
2.  outLayers = [layers[i[0] - 1] for i in net.getUnconnectedOutLayers()]
```

Pomocou metódy na prvom riadku priloženého kódu dostanem názvy všetkých vrstiev v umelej neurónovej sieti. Nepripojené a v podstate výstupne vrstvy dostaneme pomocou

metódy *getUnconnectedOutLayers()*. Výsledok uložím do lokálnej premennej *outLayers* a následne ju použijem ako parameter do funkcie *forward*.

```
1.  outs = net.forward(outLayers)
```

5.1.3 Spracovanie výstupu detekcie

Z predchádzajúcej kapitoly dostávam výstup z neurónovej siete a ostáva už len spracovanie výstupu. Lokálna premenná *outs* obsahuje dvojrozmerné pole s detekovanými objektami. Dôležite pri spracovaní výstupu je inicializácia prahovej hodnoty spoľahlivosti. Hodnota je v rozsahu od 0 po 1. Každý nájdený objekt z výstupu neurónovej siete obsahuje hodnotu spoľahlivosti pre danú triedu. Z tohto dôvodu slúži prah spoľahlivosti ako takzvaný filter, ktorý ignoruje nájdené objekty s menšou hodnotou spoľahlivosti ako nastavená hodnota prahu spoľahlivosti. V mojom prípade som ju nastavil na hodnotu 0.6, pretože detekované objekty, ktorých hodnota spoľahlivosti je menšia ako 0.6, môžu byť nesprávne detekované a tým ovplyvniť výsledky pri hľadaní nového ohniskového bodu.

```
1.  minThreshold = 0.6
```

Nájdená detekcia okrem hodnoty spoľahlivosti obsahuje aj ďalšie dôležité hodnoty. Medzi tieto hodnoty patria *x* a *y* súradnice stredu pre nájdený objekt z daného vstupného obrázka. Ďalej sem patria hodnoty predstavujúce šírku a výšku obrázka. Všetky hodnoty sú normalizované a preto ich musím previesť na skutočné hodnoty vynásobením príslušnou hodnotou výšky a šírky daného vstupného obrázka.

```
1.  center_x = int(detection[0] * frameWidth)
2.  center_y = int(detection[1] * frameHeight)
3.  width = int(detection[2] * frameWidth)
4.  height = int(detection[3] * frameHeight)
5.  left = int(center_x - width / 2)
6.  top = int(center_y - height / 2)
7.  resultsObjects = detection[5:]
8.  classId = np.argmax(resultsObjects)
9.  confidence = resultsObjects[classId]
```

V tomto kroku už máme všetky potrebné informácie o nájdených objektoch z obrázka. V prípade ak je viacej objektov na obrázku nastáva jav prekrývania, že na výstupe dostávam pre jeden objekt viacej ohraničujúcich rámikov. Preto vykonám potlačenie ohraničujúcich rámikov(non maximum suppression) pomocou funkcie z DNN modulu.

```
1.  indexes = cv.dnn.NMSBoxes(boxes, confidences, minThreshold, nmsThreshold)
```

Prvým parametrom pre funkciu *NMSBoxes* je pole nájdených objektov s ich súradnicami a veľkosťou. Druhý parameter obsahuje pole spoľahlivosti pre nájdené objekty. Tretí parameter je nastavená hodnota prahu spoľahlivosti. Posledný parameter je nastaviteľná hodnota potláčania ohraničujúcich rámikov. Hodnota je v rozmedzí od 0 po 1. Ak nastavím hodnotu napríklad 0,1 prekrývajúce objekty nebudú detekované. Naopak ak túto hodnotu nastavím na vysoké číslo napríklad 0.9 dostanem niekoľko ohraničujúcich rámikov pre jeden detekovaný objekt. Preto som zvolil hodnotu 0,4. Všetky objekty a ich príslušné parametre si uložím do poľa objektov. Trieda *detect* je definovaná nasledovne.

```
1.  def __init__(self, title, importance, confidence, x, y, width, height):
2.      self.title = title
3.      self.importance = importance
4.      self.confidence = confidence
5.      self.x = x
6.      self.y = y
7.      self.width = width
8.      self.height = height
```

Obr. 26. Ukážka detekcie objektov.

5.1.4 Hľadanie ohniskového bodu

Ako som už spomínal dôležitým faktorom pri orezávaní a vytváraní náhľadov pre daný obrázok je určiť ohniskový bod. Na určenie ohniskového bodu použijem nájdene objekty z predchádzajúcej kapitoly. Používam dve možnosti na nájdenie ohniskového bodu.

Prvá možnosť sa zameriava na zoskupenie všetkých objektov a nájdenie stredu týchto bodov bez ohľadu na dôležitosť danej triedy objektu.

```
1.  xObject = min(self.detectObjects, key=itemgetter('x'))
2.  yObject = min(self.detectObjects, key=itemgetter('y'))
3.  right, bottom = 0, 0
4.
5.  for object in self.detectObjects:
6.      if object.x + object.width > right:
7.          right = object.x + object.width
8.      if object.y + object.height > bottom:
9.          bottom = object.y + object.height
10.     return Rect(xObject.x - self.padding,
11.                 yObject.y - self.padding,
12.                 (right - xObject.x) + self.padding * 2,
13.                 (bottom - yObject.y) + self.padding * 2)
```

Výsledkom z danej metódy je zoskupený jeden veľký objekt, ktorý obsahuje aj súradnicu stredu.

Obr. 27. Ukážka zoskupenia objektov a ohniskového bodu.

Druhá možnosť, ktorú som navrhol a implementoval je zoskupenie iba dôležitejších objektov tried. Každej triede, ktorú je schopná neurónová sieť detekovať, som priradil hodnotu dôležitosti. Napríklad trieda osoba má vyššiu hodnotu dôležitosti ako napríklad trieda nožničky. Samozrejme v tomto štádiu som si určil hodnoty dôležitosti subjektívne, ale v prípade

potreby je možnosť ich upraviť. Pri tejto možnosti je veľmi dôležitý parameter, ktorý obsahuje minimálnu hodnotu dôležitosti. Napríklad ak chcem zoskupiť a použiť iba dôležité triedy ako je osoba nastavím túto hodnotu na 10. Implementácia oproti prvej možnosti je iba rozšírená o filter dôležitosti nasledovne.

```
1.  self.detectObjects = [object for object in detectObjects if object.importance >= minImportance]
```

V tomto prípade dostávam súradnice ohniskového bodu len z objektov, ktoré spĺňajú hodnotu dôležitosti.

5.2 Prepojenie detektoru pomocou API

Momentálne detektor funguje len lokálne a je možné ho spustiť len pomocou príkazového riadku. Na to aby som vytvoril komplexné riešenie som musel vytvoriť rozhranie API. API bude fungovať ako prepojenie medzi webovým riešením a detektorom využívajúc REST architektúru. Aby som zachoval konzistentnosť rozhodol som sa použiť webový framework Flask napísaný v Pythone. Flask je klasifikovaný ako mikro framework pretože nevyžaduje žiadne prídavné nástroje alebo knižnice. API bude komunikovať s webovou aplikáciou pomocou HTTP požiadaviek a ako výstup som rozhodol použiť formát JSON kvôli jednoduchosti spracovania. [24]

5.2.1 Dokumentácia API služieb

Popis: Služba vracia pole nájdených objektov vo formáte JSON.

URL: <server>/smartc/detections

Metóda: POST

Telo požiadavky: imageUrl: povinný parameter

```
1.  {
2.      "imageUrl": "http://smartc.dsoft.bid/2782381_ORIG.jpg"
3.  }
```

Odpoveď: 200 – všetko prebehlo v poriadku, 500 – serverová chyba

```
1.  {"objects":
2.      [{"title": "person",
3.         "importance": 10,
4.         "confidence": 0.996971070766449,
5.         "x": 559,
6.         "y": 475,
7.         "width": 107,
8.         "height": 174}]
9.  }
```

Popis: Služba vracia súradnice zoskupeného objektu a ohniskového bodu

URL: <server>/smartc/groupObject

Metóda: POST

Telo požiadavky: imageUrl: povinný parameter, minImportance: voliteľný parameter

```
{
    "imageUrl": "http://smartc.dsoft.bid/2782381_ORIG.jpg",
    "minImportance": 8
}
```

Odpoveď: 200 – všetko prebehlo v poriadku, 500 – serverová chyba

```
{
    "groupObject": {
        "x": 218,
        "y": 368,
        "width": 603,
        "height": 484,
        "focalPoint": {
            "x": 519.5,
            "y": 610.0
        }
    }
}
```

Popis: Služba vracia odkaz na obrázok s vyznačenými ohraničenými rámikmi

URL: <server>/smartc/detectionBoxes

Metóda: POST

Telo požiadavky: imageUrl: povinný parameter

```
{
    "imageUrl": "http://smartc.dsoft.bid/2782381_ORIG.jpg"
}
```

Odpoveď: 200 – všetko prebehlo v poriadku, 500 – serverová chyba

```
{
    "outputImageUrl": "<server>/smartc/output/2782381_ORIG_boxes.jpg"
}
```

Popis: Služba vracia odkaz na obrázok s požadovanými rozmermi vygenerovaný pomocou ohniskového bodu.

URL: <server>/smartc/cropping

Metóda: POST

Telo požiadavky:

- imageUrl: povinný parameter
- width: povinný parameter(šírka požadovaného výstupného obrázka)
- height: povinný parameter(výška požadovaného výstupného obrázka)
- minImportance: voliteľný parameter

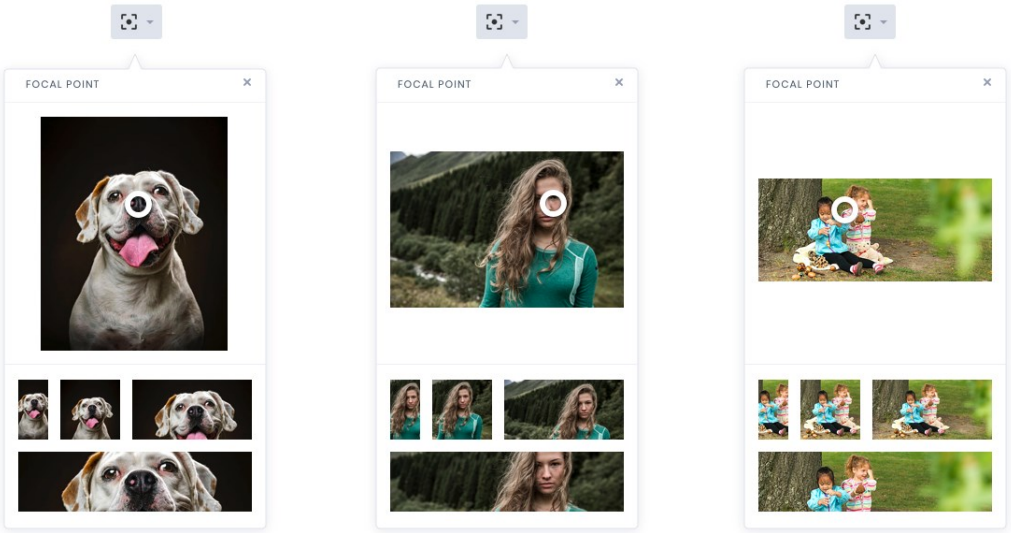
```
1. {
2.     "imageUrl": "http://smartc.dsoft.bid/2782381_ORIG.jpg",
3.     "width": 600.0,
4.     "height": 200.0
5. }
```

Odpoveď: 200 – všetko prebehlo v poriadku, 500 – serverová chyba

```
4. {
5.     "outputImageUrl": "<server>/smartc/output/2782381_ORIG_600_200.jpg"
6. }
```

5.3 Webová aplikácia

Ďalšou požiadavkou komplexného riešenia bolo navrhnúť a naprogramovať webové riešenie, ktoré sa použije v prípade ak sa nenašli žiadne objekty na obrázku alebo návrh ohniskového bodu nie je správny. Táto situácia môže nastať ak používateľ zadal abstraktný vstupný obrázok alebo je veľmi špecifický a neurčitý. Ďalším prípadom môže byť situácia ak používateľ chce náhľadové obrázky vygenerovať pomocou špecifického objektu.

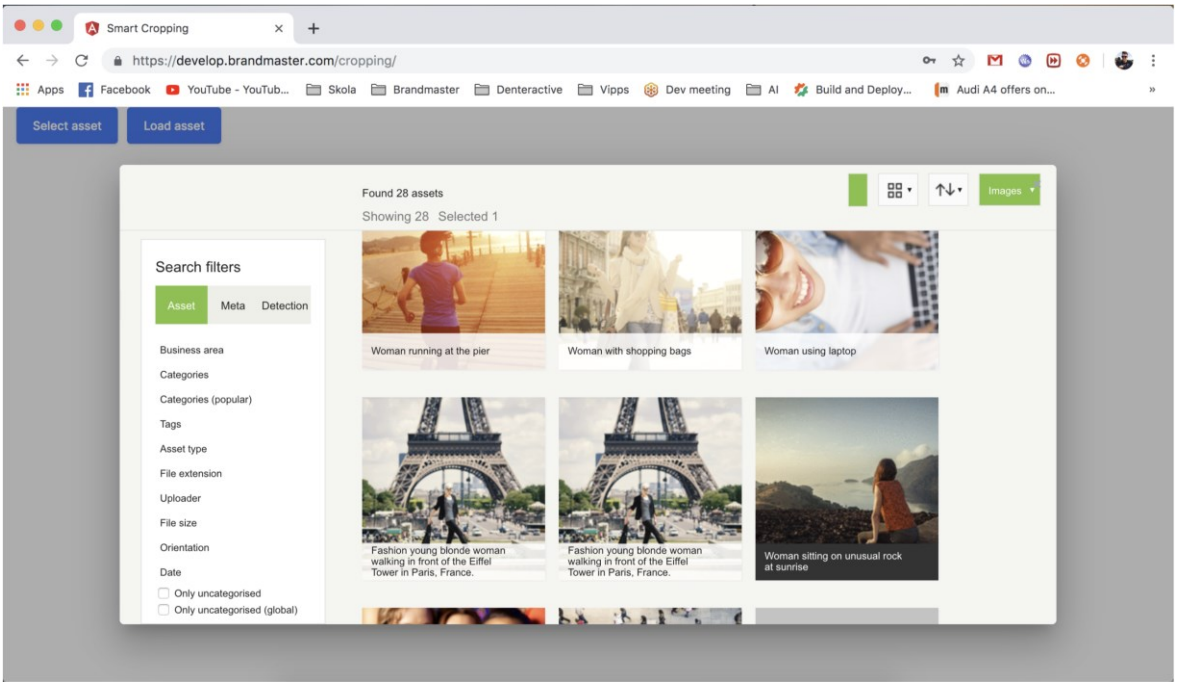


Obr. 28. Grafický návrh nastavovania ohniskového bodu.

Na vývoj webovej aplikácie som si vybral programovací jazyk TypeScript, ktorý vyvíja a udržuje spoločnosť Microsoft. Ako podporu pre programovací jazyk TypeScript som zvolil veľmi populárny open-source webový framework Angular. Túto kombináciu som si vybral kvôli predošlej znalosti a jednoduchosti používania. Použil som verziu Angularu 7.0.6. Webová aplikácia komunikuje pomocou API s multimediálnou knižnicou DAM. Z tohto dôvodu sa všetky akcie vykonané používateľom ukladajú a posielajú do multimediálnej knižnice [26].

5.3.1 Implementácia webovej aplikácie

Webovú aplikáciu som vytvoril aby používateľ mal možnosť upravovať ohniskový bod pre zvolený obrázok z mediálnej knižnice. Čo sa týka vizuálnej stránky aplikácie je veľmi jednoduchá. Po kliknutí na tlačidlo „Select asset“ sa vytvorí pripojenie s DAM knižnicou a otvorí sa modálne okno na výber obrázku. Na vykreslenie knižnice sa používa *iframe* element v HTML. Tento element sa používa na vloženie iného dokumentu do súčasného. V mojom prípade používam dokument mediálnej knižnici v mojom webovom riešení.



Obr. 29. Výber obrázku z mediálnej knižnice DAM.

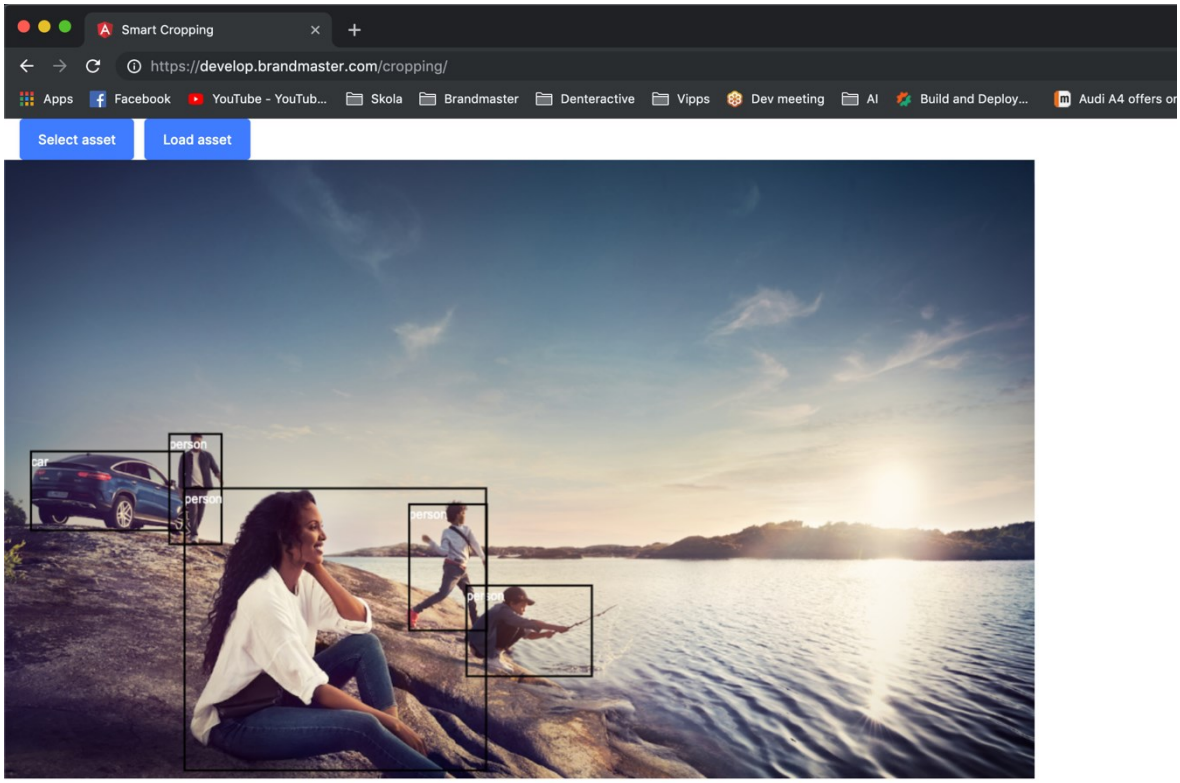
Po výbere obrázku sa automaticky zavolá požiadavka na API, ktoré som implementoval v predchádzajúcich kapitolách. Na komunikáciu s API používam štandardnú *HttpClient* knižnicu.

```
const imageData = {'imageUrl': this.image.src};
this.http.post<any>('/smartc/detections',
JSON.stringify(imageData),
AppComponent.getDefaultJsonContentHeader()).subscribe(data => {
    this.drawDetections(data.objects);
});
```

Po získaní výsledku z požiadavky dostávam pole nájdených objektov s ich súradnicami vo formáte JSON. Výsledok spracujem a vykreslím ohraničené rámiky na obrázok. Na vykreslenie používam element z HTML 5 *canvas*. Element sa používa na kreslenie grafických tvarov na webových stránkach. Nasledujúci kód vykreslí ohraničujúce rámiky pre každý objekt na obrázku s ich popisom.

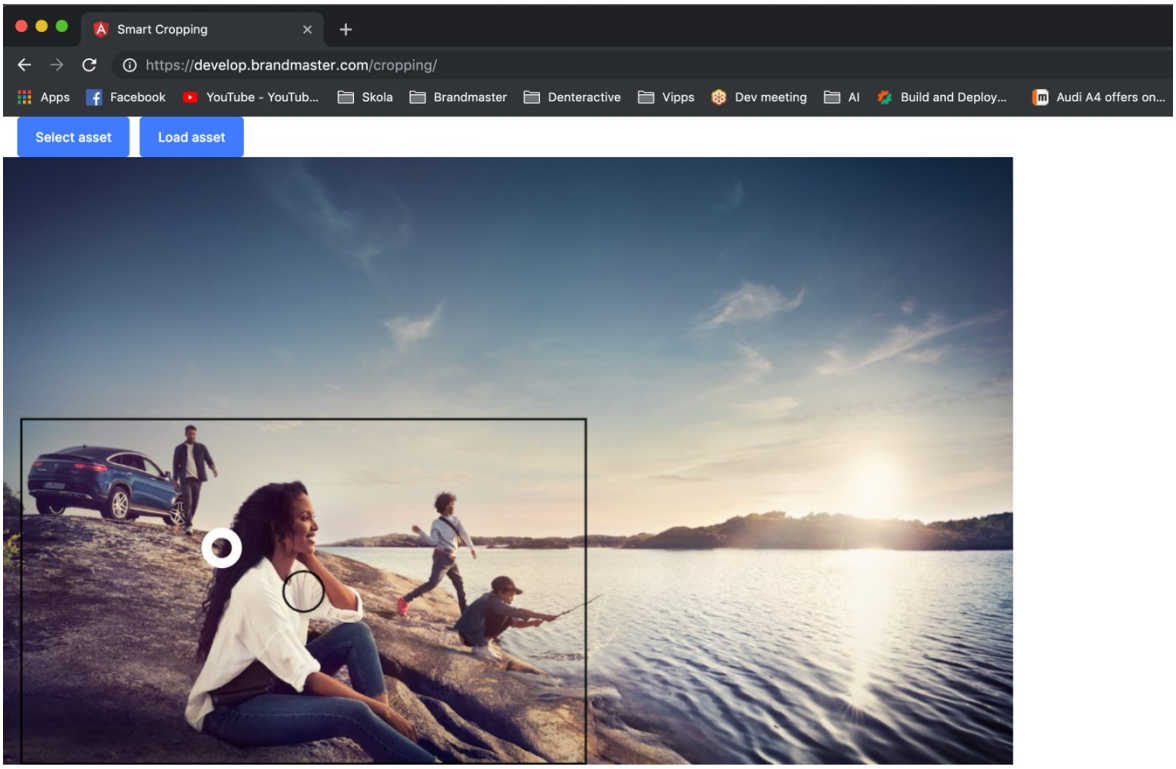
```
const detectionsCanvas = document.getElementById('detections') as HTMLCanvasEle-
ment;
detectionsCanvas.width = this.image.width;
detectionsCanvas.height = this.image.height;
const context = detectionsCanvas.getContext('2d');
context.drawImage(this.image, 0, 0, this.image.width, this.image.height);
detects.forEach((detect) => {
    context.font = '12px Arial';
    context.fillStyle = '#ffffff';
    context.fillText(detect.title, detect.x * this.imageScaling, (de-
tect.y * this.imageScaling) + (20 * this.imageScaling));
    context.beginPath();
```

```
11.     context.rect(detect.x * this.imageScaling, detect.y * this.imageScaling, de-
    tect.width * this.imageScaling, detect.height * this.imageScaling);
12.     context.lineWidth = 2;
13.     context.strokeStyle = 'black';
14.     context.stroke();
15. });
```



Obr. 30. Vykreslenie ohraničujúcich rámikov vo webovej aplikácii.

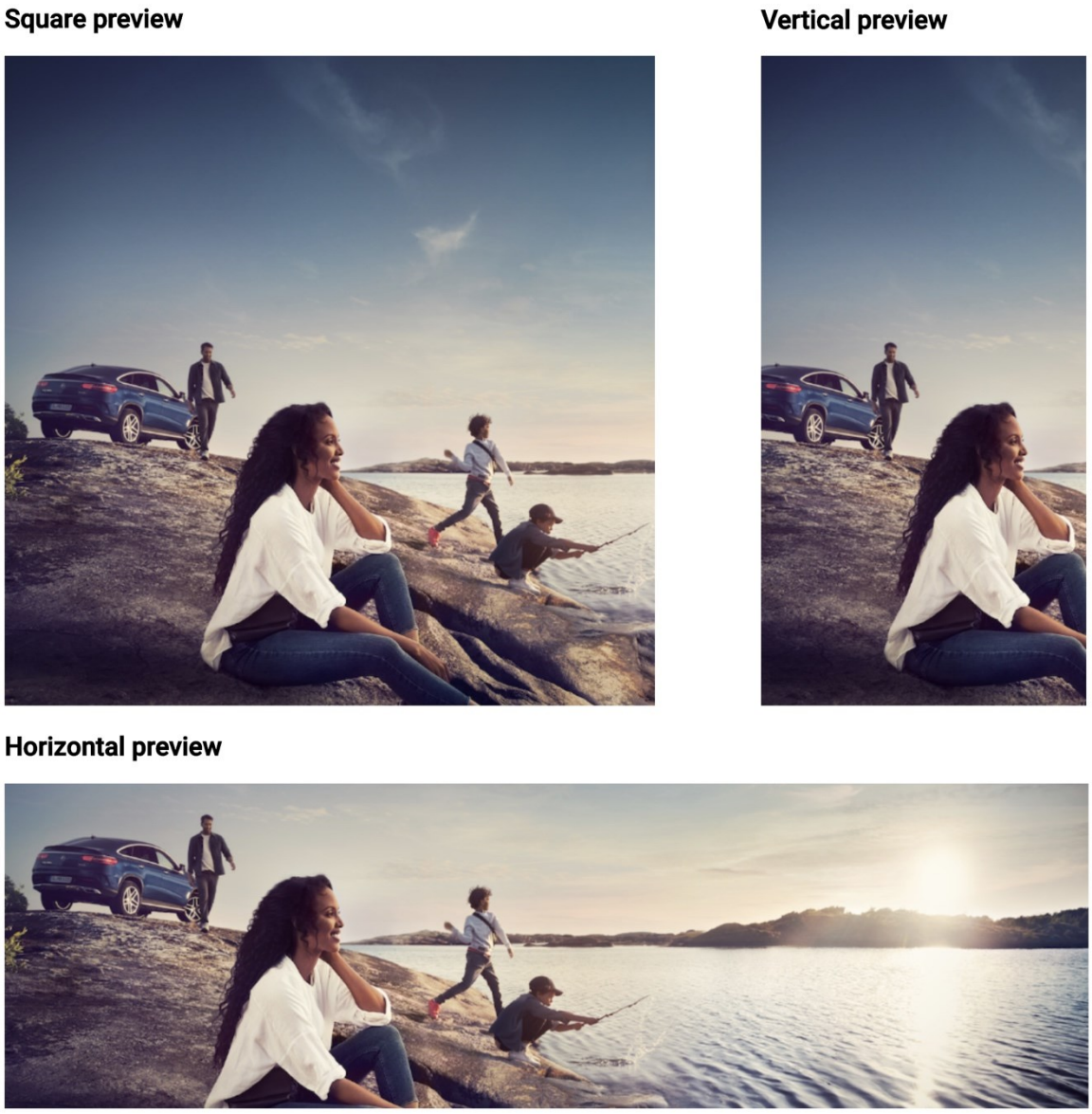
Pomocou ďalšej služby */smartc/groupObject* z API dostávam súradnice ohniskového bodu a zoskupeného objektu. Volanie služby je veľmi podobné ako v predchádzajúcom prípade. Aby som uľahčil manipuláciu pre používateľov pri nastavovaní ohniskového bodu použil som funkciu ťahaj a pusť(drag and drop). Danú funkciu som implementoval pomocou knižnice PlainDraggable. Ako predvolené nastavenie ohniskového bodu sa použije navrhovaný bod z API. Pri manipulácii s ohniskovým bodom sa automaticky prekresľujú náhľady v štvorcom, horizontálnom a vertikálnom formáte kvôli vizualizácii a kontrole. Následne po kliknutí na tlačidlo „Save“ sa odošle požiadavka na mediálnu knižnicu o zmene ohniskového bodu.



Obr. 31. Ukážka manipulácie s ohniskovým bodom.

Pri generovaní náhľadov podľa ohniskového bodu používam taktiež *canvas* element, kvôli zachovaniu rozlíšenia obrázkov k pomeru rozlíšenia monitoru. Na nasledujúcom kódu je implementácia generickej funkcie na generovanie náhľadov.

```
1.  private generatePreview(elementId, focalPoint) {
2.      const previewHorizontalCanvas = document.getElementById(elementId) as HTMLCanvasElement;
3.      const context = previewHorizontalCanvas.getContext('2d');
4.      const isHorizontal = previewHorizontalCanvas.width > previewHorizontalCanvas.height;
5.      const scaling = isHorizontal ? previewHorizontalCanvas.width / this.image.width : previewHor-
    izontalCanvas.height / this.image.height;
6.
7.      const scaledImageWidth = this.image.width * scaling;
8.      const scaledImageHeight = this.image.height * scaling;
9.      const scaledFocalPointY = focalPoint.y * scaling;
10.     const scaledFocalPointX = focalPoint.x * scaling;
11.
12.     const focalOffset = isHorizontal ? scaledImageHeight / 2 - scaledFocalPointY : scaledImage-
    Width / 2 - scaledFocalPointX;
13.     const diff = isHorizontal ? (scaledImageHeight - previewHorizontalCanvas.height) / 2 : (scal-
    edImageWidth - previewHorizontalCanvas.width) / 2;
14.     const diffCanvasWithImage = isHorizontal ? previewHorizontalCanvas.height - scaledImage-
    Height : previewHorizontalCanvas.width - scaledImageWidth;
15.     let mainCoordinate = - (diff - focalOffset);
16.     if (mainCoordinate > 0) {
17.       mainCoordinate = 0;
18.     } else if (mainCoordinate < diffCanvasWithImage) {
19.       mainCoordinate = diffCanvasWithImage;
20.     }
21.     context.drawImage(this.image, isHorizontal ? 0 : mainCoordinate, isHorizontal ? mainCoordi-
    nate : 0, scaledImageWidth, scaledImageHeight);
22. }
```



Obr. 32. Ukážka náhľadov pri zmene ohniskového bodu.

5.4 Nasadenie na server pomocou nástroja Docker

Nemenej dôležitá časť celého riešenia je nasadenie na reálny server. V mojom prípade som si zvolil Linux server a program Docker. Docker je platforma pre vývojárov a adminov na vývoj, nasadzovanie a spúšťanie aplikácií pomocou kontajnerov. Samotné kontajnery nie sú nové, ale ich použitie na ľahké nasadenie aplikácií je. Môžeme to chápať ako nástroj, ktorý spustí aplikáciu, ale s minimálnymi požiadavkami na konfiguráciu a správu. V mojom prípade ak vyvíjam aplikáciu na rôznych platformách a konfiguráciách stačí si spustiť pre každú

aplikáciu samostatný kontajner. Docker kontajner som použil pri nasadzovaní API a detektora. Inštalácia Docker nástroja na Linux serveri je veľmi jednoduchá a robí sa nasledovným príkazom [25].

```
1. sudo apt-get install docker-engine -y
```

Po nastavení servera som musel nastaviť Docker konfiguráciu pre môj projekt. Vytvoril som si súbor *requirements.txt*, ktorý obsahuje zoznam použitých knižníc v projekte.

```
1. flask
2. numpy
3. awscli
4. opencv-python
```

V ďalšom kroku konfigurácie je nutné vytvoriť súbor *Dockerfile*. Dockerfile je jednoduchý textový súbor, ktorý popisuje, ako má vypadať výsledný obraz projektu. V podstate je to náhrada za kroky, ktoré sa robia bežne pri konfigurácii projektu na nový server. Dockerfile pre API a detektor vyzerá nasledovne.

```
1. FROM python:3.7
2.
3. RUN apt-get update \
4.     && apt-get install -y \
5.         build-essential \
6.         cmake \
7.         git \
8.         wget \
9.         unzip \
10.        yasm \
11.        pkg-config \
12.        libswscale-dev \
13.        libtbb2 \
14.        libtbb-dev \
15.        libjpeg-dev \
16.        libpng-dev \
17.        libtiff-dev \
18.        libavformat-dev \
19.        libpq-dev \
20.    && rm -rf /var/lib/apt/lists/*
21.
22. ADD . /app
23. WORKDIR /app
24. RUN pip install -r requirements.txt
25. CMD python src/app.py
```

Následne pomocou tohto súboru som schopný vybuildovať Docker image. V mojom prípade som navyše pridal súbor docker-compose.yml, kvôli budúcej expanzii projektu a podpore viacerých kontajnerov spustených v jeden čas. V momentálnom štádiu tento súbor obsahuje iba konfiguráciu Flask servera, ktorý beží na porte 5000.

```
1. version: '2'
```

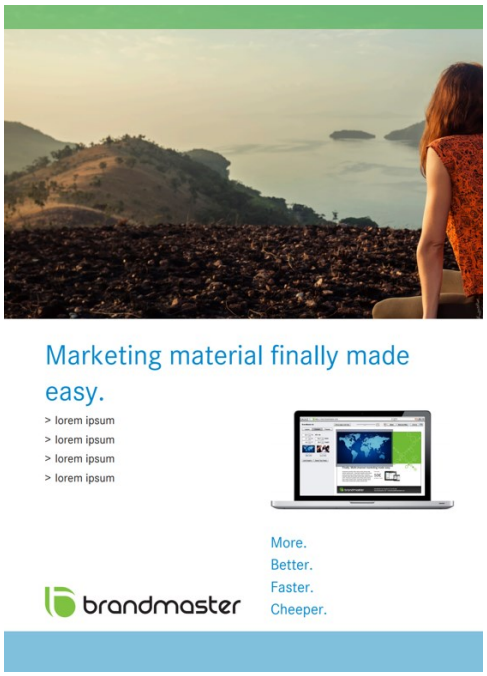
```
services:
    web:
        build: .
        ports:
            - "5000:5000"
        volumes:
            - .:/app
```

Po nastavení konfiguračných súborov requirements.txt, Dockerfile a docker-compose.yml môžem spustiť server jednoduchým príkazom v príkazovom riadku.

```
docker-compose up
```

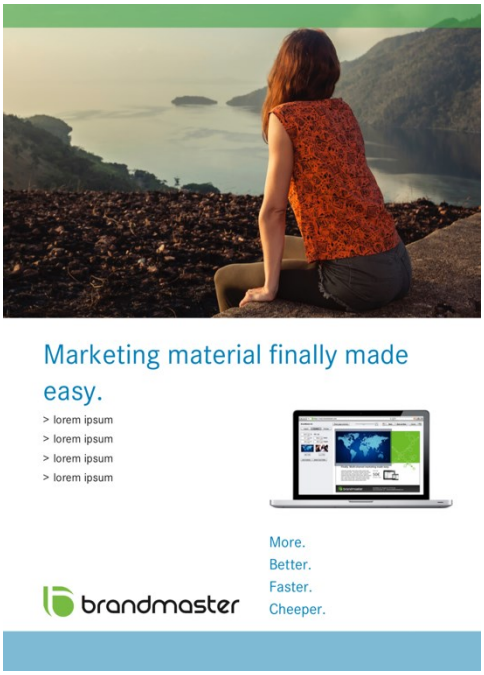
5.5 Použitie riešenia

V momentálnom štádiu realizácie sa používa moje riešenie pri výrobe marketingových materiálov. Zatiaľ len pre testovacie a interné účely. Na demonštráciu riešenia som použil obrázok, kde hlavný objekt sa nachádza na okraji obrázku. Obrázok som následne vložil do hlavičky testovacieho marketingového materiálu. Na obr. 33 je použité klasické centrovanie na stred obrázka bez pomoci detekovaných objektov. Je možné vidieť, že hlavný objekt je odseknutý.



Obr. 33. Marketingový materiál bez využitia detekcie objektov.

Pri rovnakom testovacom materiáli ale s použitím riešenia na detekciu objektov je možné automaticky vygenerovať nasledovný marketingový materiál bez manuálneho zásahu používateľom. A výsledok na obr. 34 je podstatne lepší a použiteľnejší.



Obr. 34. Marketingový materiál s využitím detekcie objektov.

ZÁVER

Cieľom diplomovej bolo navrhnúť a implementovať nový spôsob orezávania a generovania náhľadov z obrázkov, ktoré sa ďalej používajú pri generovaní marketingových materiálov. Ako riešenie pri hľadaní ohniskového bodu na obrázku som si zvolil umelú inteligenciu a neurónové siete. Použil som algoritmus na detekciu objektov a podľa nájdených objektov som označil ohniskový bod podľa ktorého sa orezávajú náhľady.

V teoretickej časti práce sa nachádzajú znalosti o mediálnej knižnici do ktorej sme riešenie implementovali. Teoretická časť taktiež popisuje princíp umelej inteligencie, neurónových sietí, konvolučných sietí a metód detekcie objektov. Posledná časť teoretickej časti sa zameriava na popis knižnice použitej pri spracovaní obrazu OpenCV. Popis knižnice obsahuje modul na podporu a implementáciu hlbokých neurónových sieti.

V praktickej časti som sa venoval návrhu a implementácii metód na orezávanie náhľadov pre obrázky z mediálnej knižnice. Implementácia pozostáva z troch častí. V prvej časti je implementovaný algoritmus na detekciu objektov z obrázkov. V druhej časti som vytvoril API prepojenie na komunikáciu medzi webovou aplikáciou a aplikáciou na detekovanie objektov. V tretej časti som graficky navrhol a následne implementoval webové riešenie na načítanie a uloženie nastavení obrázka z mediálnej knižnice. Pre zvolený obrázok sa automaticky pomocou API spustí detekcia objektov, navrhne sa ohniskový bod a vygenerujú sa náhľady. Používateľ má možnosť manuálne nastaviť ohniskový bod pomocou intuitívneho ovládania, čo spustí nový proces generovania náhľadov.

Projekt je v rannom štádiu realizácie. Momentálne je nasadený na vývojárskom serveri a používajú ho zväčša interní zamestnanci spoločnosti. Po prvom mesiace používania je odozva veľmi dobrá a vygenerované náhľady sa používajú pri testovacích alebo prezentačných marketingových materiáloch.

ZOZNAM POUŽITEJ LITERATÚRY

[1] Media Bank – DAM. BrandMaster AS [online]. [cit. 2019-03-20]. Dostupné z: https://www.brandmaster.com/products/media-bank-dam/

[2] Vision AI. Google Cloud [online]. [cit. 2019-04-02]. Dostupné z: https://cloud.google.com/vision/

[3] H. MOOR, James. The Turing Test: The Elusive Standard of Artificial Intelligence. Dartmouth College, Hanover, USA: Springer, Dordrecht, 2003. ISBN 978-94-010-0105-2.

[4] ZELINKA, Ivan.: Umělá inteligence I. Volume 1. Zlín, Vutium, Brno, 1998, 126 p. ISBN 80-214-1163-5.

[5] BÍLA, Jiří. Umělá inteligence a neuronové sítě v aplikacích. Vyd. 1. Praha: Vydavatelství ČVUT, 1995, 115 s. ISBN 80-010-1275-1.

[6] ŠÍMA, Jiří a Neruda R. Teoretické otázky neuronových sítí. Vyd. 1.. Praha: MATFYZ press, 1996, 390 s. ISBN 80-858-6318-9.

[7] NÁVRAT, Pavol, Mária BIELIKOVÁ a Ľubica BEŇUŠKOVÁ. Umelá inteligencia. Vazovova 5, Bratislava: Vydavateľstvo STU, 2002, 396 s. ISBN 80-227-1645-6.

[8] ŠIKUDOVÁ, Elena. Počítačové videnie: detekcia a rozpoznávanie objektov. Praha: Wikina, [2014]. ISBN 978-80-87925-06-5.

[9] RUSSELL, Stuart J., Peter NORVIG a Ernest DAVIS. Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River: Prentice Hall, c2010. ISBN 978-0-13-604259-4.

[10] ROJAS, Raul. The Backpropagation Algorithm [online]. In: . Berlin: Springer-Verlag, 1996, s. 184 [cit. 2019-03-25]. Dostupné z: https://page.mi.fu-berlin.de/rojas/neural/chapter/K7.pdf

[11] LI, Baoxin. Convolutional neural networks in visual computing: A concise guide. Boca Raton: CRC Press, [2017]. ISBN 978-149-8770-392.

[12] ZHAO, Zhong-Qiu. Object Detection with Deep Learning: A Review [online]. In: . 2019, 16.4.2019, s. 21 [cit. 2019-04-21]. Dostupné z: https://arxiv.org/pdf/1807.05511.pdf

[13] GIRSHICK, Ross. Rich feature hierarchies for accurate object detection and semantic segmentation [online]. In: . UC Berkeley, 2014, Tech report (v5) [cit. 2019-03-09]. Dostupné z: https://arxiv.org/pdf/1311.2524.pdf

[14 GIRSHICK, Ross. Fast R-CNN [online]. In: . 2015 [cit. 2019-03-09]. Dostupné z: https://arxiv.org/pdf/1504.08083.pdf

[15] GIRSHICK, Ross. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks [online]. In: . 2016 [cit. 2019-03-09]. Dostupné z: https://arxiv.org/pdf/1506.01497.pdf

[16] SZEGEDY, Christian. SSD: Single Shot MultiBox Detector [online]. In: . 2016 [cit. 2019-03-09]. Dostupné z: https://arxiv.org/pdf/1512.02325.pdf

[17] REDMON, Joseph. You Only Look Once: Unified, Real-Time Object Detection [online]. In: . 2016 [cit. 2019-03-09]. Dostupné z: https://arxiv.org/pdf/1512.02325.pdf

[18] GOLDSBOROUGH, Peter. A Tour of TensorFlow [online]. In: . 2016 [cit. 2019-03-16]. Dostupné z: https://arxiv.org/pdf/1610.01178.pdf

[19] REDMON, Joseph. Darknet: Open Source Neural Networks in C [online]. In: . 2013--2016 [cit. 2019-04-02]. Dostupné z: http://pjreddie.com/darknet/

[20] DADHICH, Abhinav. Practical Computer Vision. Packt Publishing, 2018, 234 s. ISBN 9781788297684.

[21] Deep Learning in OpenCV [online]. [cit. 2019-04-18]. Dostupné z: https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV

[22] RYBNIKOV, Alexander. Deep Learning, теперь и в OpenCV [online]. In: . 20.07.2017 [cit. 2019-04-10]. Dostupné z: https://habr.com/en/company/intel/blog/333612/

[23] REDMON, Joseph a Ali FARHADI. YOLOv3: An Incremental Improvement [online]. In: . arXiv, 2018, 08.04.2018 [cit. 2019-04-22]. Dostupné z: https://arxiv.org/pdf/1804.02767.pdf

[24] Flask (A Python Microframework) [online]. [cit. 2019-04-02]. Dostupné z: http://flask.pocoo.org/

[25] Enterprise Application Container Platform | Docker [online]. [cit. 2019-04-15]. Dostupné z: https://www.docker.com

[26] TypeScript - JavaScript that scales [online]. [cit. 2019-04-11]. Dostupné z: https://www.typescriptlang.org/

[27] HUI, Jonathan. MAP (mean Average Precision) for Object Detection [online]. In: . 07.03.2018 [cit. 2019-03-25]. Dostupné z: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

[28] File management, metadata integration | Adobe Extensible Metadata Platform (XMP) [online]. [cit. 2019-05-03]. Dostupné z: https://www.adobe.com/products/xmp.html

[29] BRITZ, Denny. Understanding Convolutional Neural Networks for NLP [online]. [cit. 2019-04-06]. Dostupné z: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[30] An overview of HTTP [online]. [cit. 2019-04-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[31] What is REST [online]. [cit. 2019-04-20]. Dostupné z: https://restfulapi.net/

[32] The PASCAL Visual Object Classes Homepage [online]. [cit. 2019-05-07]. Dostupné z: http://host.robots.ox.ac.uk/pascal/VOC/

[33] COCO - Common Objects in Context [online]. [cit. 2019-05-07]. Dostupné z: http://cocodataset.org

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

HTTP Hypertext Transfer Protocol.

REST Representation state transfer.

API Application programming interface.

DAM Digital Asset Management.

CNN Convolutional Neural Network.

MSE Mean Square Error.

SVM Support Vector Machines.

ROI Region of interest.

TPU Tensor Processing Unit.

XMP Extensible Metadata Platform.

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

PRÍLOHA P I: ELEKTRONICKÁ PRÍLOHA NA CD

Elektronická príloha na CD obsahuje samotnú prácu vo formáte pdf a kompletné zdrojové kódy.

PRÍLOHA P II: PRÍKLADY VYGENEROVANÝCH NÁHĽADOV

Pre nasledujúci zdrojový obrázok (Obr. 35) systém mal za úlohu vygenerovať štvorcový náhľadový obrázok.

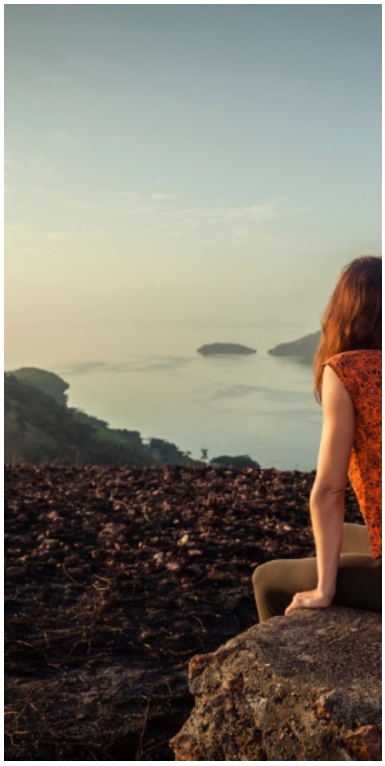
Obr. 35. Zdrojový obrázok pre štvorcový náhľad.

Obr. 36. Štvorcový náhľad použitím bežného spôsobu so stredovým ohniskovým bodom.

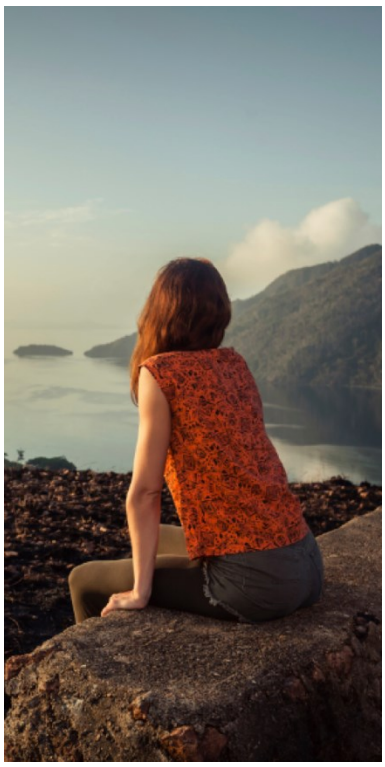
Obr. 37. Štvorcový náhľad vygenerovaný použitím navrhovaného ohniskového bodu z neurónovej siete pre obr.35.

V ďalšom prípade mal systém vygenerovať vertikálny náhľadový obrázok pre zdrojový obrázok (Obr. 38.).

Obr. 38. Zdrojový obrázok pre vertikálny náhľad.



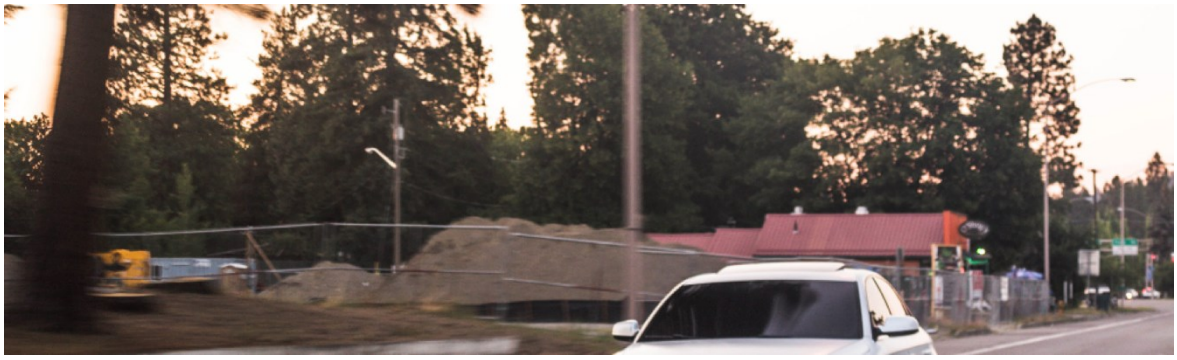
Obr. 39. Vertikálny náhľad použitím bežného spôsobu so stredovým ohniskovým bodom.



Obr. 40. Vertikálny náhľad vygenerovaný použitím navrhovaného ohniskového bodu z neurónovej siete pre obr.38.

V poslednom demonštračnom príklade mal systém vygenerovať horizontálny náhľadový obrázok pre zdrojový obrázok (Obr.41).

Obr. 41. Zdrojový obrázok pre horizontálny náhľad.



Obr. 42. Horizontálny náhľad použitím bežného spôsobu so stredovým ohniskovým bodom.

Obr. 43. Horizontálny náhľad vygenerovaný použitím navrhovaného ohniskového bodu z neurónovej siete pre obr.41.